Escuela
Politécnica
Superior

# MureTools

## An Optical Music Recognition Supporting System

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Javier Martínez Segura
Tutores:
Jose Manuel Iñesta Quereda
Antonio Ríos Vila

Julio 2021

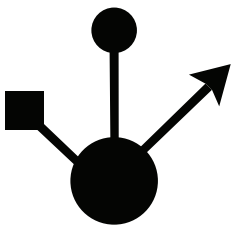
Universitat d'Alacant
Universidad de Alicante

# MureTools

**An Optical Music Recognition Supporting System**

**Autor**
Javier Martínez Segura

**Tutores**
Jose Manuel Iñesta Quereda
*Departamento de Lenguajes y Sistemas Informáticos*
Antonio Ríos Vila
*Departamento de Lenguajes y Sistemas Informáticos*



Grado en Ingeniería Multimedia

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2021

# Motivation, justification and general purpose

Machine learning and deep learning as a whole has progressed and ingrained more and more in our society in recent years. Many of the tasks within our daily lives are being changed and influenced by the solutions given by these fields. This is the reason why many other fields are being approached through these solutions, and thus also having demand in order to improve or create ways of satisfying the needs we have.

Then, when it was time to choose a topic for this final undergraduate project, I did not have a clear idea about what to do it about, but I wanted to try something new and out of my comfort zone. So after hearing from some friends about this field and their explanations about what they were doing for their own undergraduate project, it sparked in me curiosity for this field. Once I decided this, I contacted my tutor, Jose Manuel, who gave me some proposals about projects within this field, also seeing that they were related to music and having a huge passion for music, I saw it as a signal and a perfect opportunity of including it within this undergraduate thesis.

So despite embarking in a totally new field with new concepts and technologies to learn about, and even though it was expected to struggle and face multiple difficulties through it, I had the security that I was going to learn more about this new world and improve as a result of this whole experience.

# Acknowledgments

First to start off I would like to thank my tutors, Jose Manuel and Antonio for giving me the opportunity of working in such a project and thus learning many new different things that otherwise I would not have had the opportunity to learn, so thanks for teaching me. Also again, want to thank Jose Manuel for the subjects taught during my degree and for all the passion shown in teaching and education in general.

Also I would like to thank all the people who have helped me and shared with me all of these years of university, and that have been with me for many years from my high school days until now so that includes my colleagues in Dune Studio: Mario, Mónica, Moisés and Martín for an incredible last year. All of my friends from high school, university and that I was able to meet until now who are: Ethan, Fran, Nico, Roque, María, Esther, Yera, Irene, José, Alberto, Dani, Andrés, Eduardo, Raquel, Vero and Iván among many others that I will always remember, thanks for walking along with me to all the classes we shared and for always being there for me.

I want to also thank all of my band mates in *The Break*: Carlos Jorge and Erik. For starting a band with me and giving me the opportunity to share one of my biggest passions in life with other people which is one of the best things that has happened in my life so far.

Last but not least, thanks to all my family for always supporting and believing in me which includes: my two sisters Mila and María José, my parents Francisca and José, my uncle José my grandmother Carmen, my cousin Alex, my brother-in-law Antonio and the rest of my relatives, thanks to all of you.

To all of you, thanks for inspiring me to become a better professional, a better musician and a better person each day.

# Abstract

This project has the intention of providing supporting tools in the OMR (Optical Music Recognition) field by easing training and evaluation tasks. Through this whole work, the OMR field along with the relevant processes being carried out with the objective of digitizing and preserving musical pieces and tradition will be presented. Additionally, not only formats and representations will be introduced, but also neural networks models proven to be effective in this field, as well as its relevant concepts and metrics will be included throughout, unveiling also the procedures carried out in order to achieve the proposed objectives, as well as the solutions for issues that arise. Furthermore it will be important to showcase the possibilities of integrating all of these kind of tasks through web microservices and synchronous and asynchronous protocols, using task queuing so processes can take place over time steadily and effectively.

# Resumen

Este proyecto tiene la intención de proporcionar herramientas de soporte en el campo del OMR (Reconocimiento Óptico de Música), facilitando el entreno y la evaluación de tareas. Mediante este trabajo, el campo del OMR junto con los procesos llevados a cabo con el objetivo de la digitalización y preservación de piezas musicales y tradición serán presentados. Adicionalmente, no solo los formatos y representaciones serán introducidos sino también los modelos de redes neuronales efectivos en este campo, así como sus conceptos y métricas relevantes serán incluidos a través de todo, desvelando los procedimientos llevados a cabo de manera que se alcancen los objetivos propuestos, así como las soluciones para los problemas que surjan. Además será importante demostrar las posibilidades de integrar este tipo de tareas a través de microservicios web y protocolos síncronos y asíncronos, usando encolado de tareas de maneras que los procesos se lleven a cabo durante el tiempo de forma continua y efectiva.

*Learn as though you would never be able to master it;*
*hold it as though you would be in fear of losing it.*

Confucius.

*We all have idols. Play like anyone you care about*
*but try to be yourself while you're doing so.*

B.B. King.

# Contents

# List of Figures

# List of Tables

# Listings

# 1 Introduction

Before going further into the explanation of MureTools and all the related points regarding its development, it is important to state all the influencing factors that resulted in this project as well as the context involving this field of study.

These days Machine Learning (ML) has been increasingly gaining more and more applications in our daily life, achieving systems that learn and provide us better results in multiple fields and industries. With the advances accomplished in hardware in recent years along with the new capability of using these to store huge amounts of data, and thus the emergence of terms like big data, have made possible to apply all of this in a way that is useful for us to build more robust and adaptive systems by analyzing this data.

There are many applications where we can see already the benefits of infrastructures delivering results based off data, as image recognition and more specifically, in this project's case, Optical Music Recognition (OMR).

As we mentioned previously, in terms of image recognition, it is important to state that there are multiples ways of exploiting this process and that we have the possibility to recognize any type of elements we would see in our daily life. When it comes to the extraction of characters and written texts we would be talking about the discipline known as Optical Character Recognition (OCR) and analogically speaking in the music field we would be satisfying the needs of digitize the different music notation. When speaking of music notation we refer to a group of writing systems like the ones we could have in normal written text, but concretely for representing music so that although this is found in a wide range as expressed in Calvo-Zaragoza et al. (2020), later on it can be visually encoded, grouped and unified with the purpose of preserving and providing them so that the musician can later perform these pieces.

For recognizing not only the notes, but also the different symbols indicating the key the piece is in, or that even modify the notes length or if it is accidental or not by increasing or decreasing the pitch of said note. All of these different functionalities are determined in the music score depending on what is the position they occupy on the staff and also the shape so we are able to identify what is the function they are performing within the piece, but also we can meet different elements found through the whole document which give us other type of information as the title of the piece or the author's name and lyrics if there are any, all of the previous mentioned being disposed in locations out from the staff and which distribution could vary from piece to piece of music.

Bearing in mind all of these factors and variables we could encounter when digitizing music scores, we might find ourselves trying to serve effective deep learning models that could be trained through these documents so that further in time we could fulfil more effectively getting systems for this digital conversion.

As a result of the previously explained scenario, there were needs and common points for a tool that could help us in this field so that we could tweak and adjust easily different models that have shown good results for these recognition applications in other contexts as handwriting or recognizing phonemes in speech audio among others and thus would be of utility in OMR.

And so, as a response to these needs and through a supporting role in the use of University of Alicante's platform MuRET, this project was born and planned to came to life.

Principally with MureTools we expect to create an application where we can integrate models specialized in the OMR field, and that at the same time we have the possibility to train with our own music scores and parameters so we can obtain the neural network that will accomplish the best results, bearing in mind that we will also be able to visualize the metrics in charts and logs so that we can conclude the best model for our needs.

Finally we will be able to save the model that obtained the best results for the music score provided, with the parameters sent to train with. And we expect to have a common place for all of these features in straightforward and simple web application with a frontend for the user to select all of the expressed parameters and data, and a backend that will receive the training request and execute it, including also the subsequent results shown and the best model stored.

Also ultimately through this project, is pursued the demonstration of showing the potential that a web application could achieve integrating deep learning and neural networks so that the tasks of deploying and obtaining efficient models as well as related jobs can be carried out more easily, comfortably and quickly, not only in this context and with this goal but also in many other different ones, as web services can always allow us to reach other kinds of possibilities that originally we are not able to exploit through automation and processing of tasks.

# 2 State of the Art

First and foremost before starting to get into the core of the project, it is important to give context to this whole project, as well as give important information to the understanding of the field we will be working and explaining ourselves in. So in this chapter we will explain multiples concepts related to the overall implementation of our application MureTools, including how establishes itself complementary to MuRET's environment, all the technologies and practices used in the field that are relevant to us, as well as the subsequent and next steps to take within this project.

## 2.1 Introduction to OMR

### 2.1.1 What is OMR?

For the sake of this project is also important to introduce the field that is being supplied with the models built within MureTools. This field is Optical Music Recognition (OMR), and as put in (Calvo-Zaragoza et al., 2020), it works in the same way that written text may serve as a precursor of speech, similar to Optical Character Recognition (OCR) technology that has enabled the automatic processing of written texts, reading music notation also invites automation. OMR covers the automation of this task of "reading" in the context of music.

So in order to build these models it is important to state the data that all of these models will be provided and how OMR obtains this data by the music encoding process to recover the musical notation and semantics from documents.

The process would start with the visual expression of the musical piece with a music notation in a document, one of the most frequently used notation system is Common Western Music Notation (CWMN, also known as modern staff notation). To get this visual representation, there are multiple steps until it gets to the definitive document as illustrated in 2.1, first the composed notes are collected, and later on all the proper conventions as defining the clef, key or the phrase marking that will make the piece easier to comprehend and understand for the musician performing it.

What is important to specify here is that when the final document is available, it is possible to not only recover the semantics representing the notes or pitches conforming the piece, but also other aspects from the music notation that are key to understand the piece and that describe the time and other contextual characteristics so that the resemblance with how originally the piece was written is as much as possible, bearing in mind that always there is a possibility of different performers and contexts that could create slight variations and disagreements although the piece is the same.

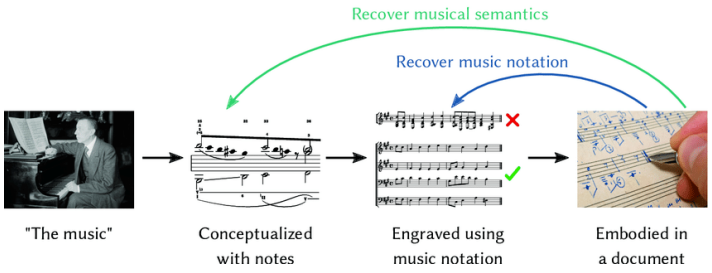
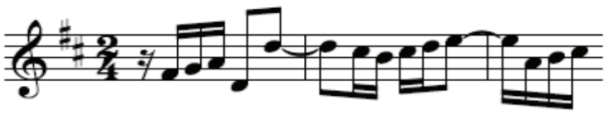



**Figure 2.1:** Pipeline followed by music until it gets represented in a document (Source: Calvo-Zaragoza et al. (2020))

### 2.1.2 Digitization, representation and formats

When digitizing this music document there are different types of files that are relevant to explain what is being employed to feed the models included in MURETOOLS.

First there are multiple options to represent musical notation digitally like MusicXML and Music Encoding Initiative (MEI) which are format widely spread and standardized, despite this, they are conceived to get the final, well-defined musical concepts and thus not really suited for the processing involved in OMR. This is why, for carrying out the conversion other formats will be taken into account, and is important to distinguish these and why they are more suited for the operation. To start, agnostic and semantic representations of a music score. The agnostic stands for the graphical information about the symbols, in other words, their shapes and positions with no musical meaning whatsoever, then the semantic representation gives it musical meaning by multiple tasks that not only give it a symbol type and a position but also a pitch within a key and an octave and its time signature among other things, understanding it better from a musical point of view, as evidenced in 2.2.



**(a)** Excerpt of a music piece

```
clef.G-L2, accidental.sharp-L5, accidental.sharp-S3, digit.2-L4, digit.4-L2, rest.sixteenth-L3,
note.beamedRight2-S1, note.beamedBoth2-L2, note.beamedLeft2-S2, note.beamedRight1-S0,
note.beamedLeft1-L4, slur.start-L4, barline-L1, slur.end-L4, note.beamedRight1-L4,
note.beamedBoth2-S3, note.beamedLeft2-L3, note.beamedRight2-S3, note.beamedBoth2-L4,
note.beamedLeft1-S4, slur.start-S4, barline-L1, slur.end-S4, note.beamedRight2-S4,
note.beamedBoth2-S2, note.beamedBoth2-L3, note.beamedLeft2-S3
```

**(b)** Representation through agnostic encoding of the excerpt

```
clef-G2, keySignature-DM, timeSignature-2/4, rest-sixteenth, note-F#4_sixteenth,
note-G4_sixteenth, note-A4_sixteenth, note-D4_eighth, note-D5_eighth, tie, barline,
note-D5_eighth, note-C#5_sixteenth, note-B4_sixteenth, note-C#5_sixteenth,
note-D5_sixteenth, note-E5_eighth, tie, barline, note-E5_sixteenth,
note-A4_sixteenth, note-B4_sixteenth, note-C#5_sixteenth
```

**(c)** Representation through semantic encoding of the excerpt

**Figure 2.2:** It is appreciated the difference between the agnostic and the semantic one as the latter is of more high level and relevant in a musical context (Source: Thomae et al. (2020))

So through these encodings, it was clearly noticeable that the relevance when it comes to OMR resides within using sequential encodings like any of the ones shown above. Addition-

ally there are other formats like **kern, that encodes only pitch and duration, plus some other common score-related information. Despite not showcasing the visual or orthographic information as said in (*Basic Notated Music / Humdrum*, n.d.), it still represents the underlying semantic information implied by a musical score, just like the semantic encoding was able to do, which also makes it viable to use it in OMR, and so it is that it will have room in MureTools. In following chapter all the formats and encodings mentioned here and important in OMR will be employed through the datasets of incipits (initial sequence of notes that identify the starting point), images and JSON used.

## 2.2 Introduction to Deep Learning

Deep Learning is included within Machine Learning as a subfield of it, due to this reason first it is important to state the definition of the latter one as the overall use of computer algorithms which provides systems the ability to automatically learn and improve through experience and data, without the need of explicit programming of said behaviour. Machine Learning as a whole gives us the tools which allow the creation of said systems that would automate many of the tasks in different fields from our daily life carrying them out even better than human themselves thanks to this ability to learn.

Deep Learning is then a type of machine learning algorithms which uses multiple layers, composed of artificial neurons that are arranged forming networks. These networks are called Artifical Neural Networks (ANN), and are a collection of units or nodes called neurons and arranged in multiple layers. So linearly these layers pass one to another the input fed to the ANN, until it reaches the final layer of output neurons as can be seen in figure 2.3, finally that will provide our predicted results by the system which could be binary (like a yes or no) or even a group of symbols (blue, red, ...).

Now in terms of explaining how neurons work (they can also be referred as a perceptron), these are no more than mathematical functions. They are conformed by weights which are depicted as the connections between the neurons and that are values that multiply the inputs provided. Once these are added together then the resulting value is passed to a non-linear function, named the activation function ($\theta$), and all of this finally eventuating in the neuron's output that will be passed on the next layer of neurons. The described basic structure of a perceptron is represented in figure 2.4.

So to sum up the functioning of these perceptrons, as we mentioned earlier they are basically mathematical functions that receive a sum of inputs. If the total value exceeds a specific threshold, then it will output a signal if the threshold is not surpassed then it won't. The input "x" is multiplied with the learned weight coefficient, and after the operations take place an output value "f(x)" is generated. So taking into account the threshold mentioned the function would be like the one shown on 2.1, which would define whether the perceptron is fired or not.

$$f(x) = \begin{cases} 1 & \text{if } \ w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Depending on which activation functions is being used there will be one or another threshold and define whether it is activated or not, a bias (b) is used, this value is also learnable like

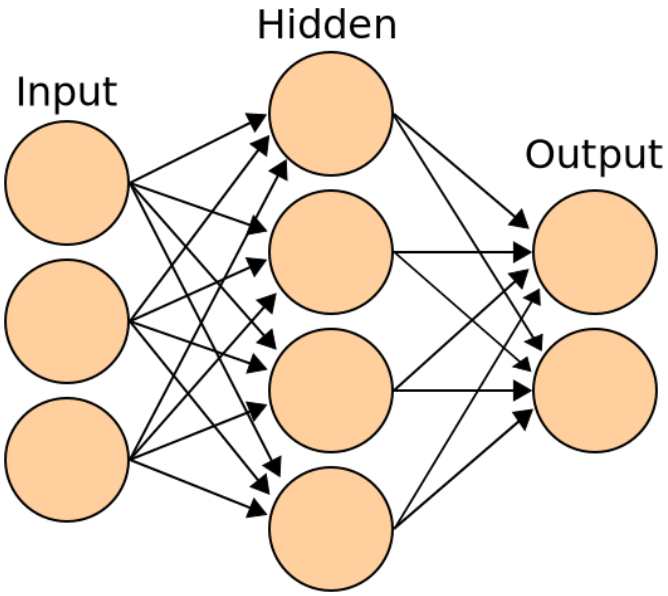



**Figure 2.3:** Architecture of an Artificial Neural Network (ANN). Image extracted from Wikimedia Commons

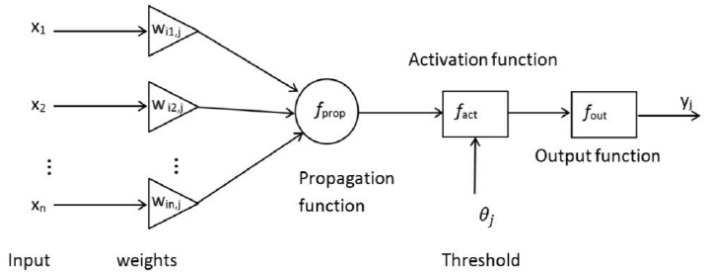



**Figure 2.4:** Structure of a binary perceptron (Source: *What is Perceptron | Simplilearn* (n.d.))

the weights (w), that is added to the total value in order to determine if the neuron's output will be propagated forward to the next layers through the network. For example, one the most used activation functions Rectified Linear Unit (ReLU), will cause the output to be comprehended between 0 and 1, imagine the output given by the operation illustrated in function 2.2 gives us a negative value for instance -0.2, thanks to the bias added (let's say 0.5) without any dependence on the input values the output results in 0.3 and the neuron which originally was not fired through these parameters, is now shifted and between the threshold established, being now considered to fire. Consequently this achieves a broader range within the network, as values that were considered not to fire the neuron are now considered to fire it.

$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right) \tag{2.2}$$

So as we have seen until now in these ANN, first the data will be fed to the input layer, which has as many neurons as data units provided, then the processing of this data begins through the multiple hidden layers until arriving to the final output layer which has as many neurons as there are categories to classify. In order to really explain how the learning takes place in the neural network, we ought to look in how the processing takes place within the hidden layers.

Through multiple iterations the previous mentioned learnable parameters (weights and bias) are adjusted and improved, since at a first instance they are initialized randomly. These parameters keep improving through iterations called epochs where values like the loss let us know how much deviation is still present in these always changing parameters. These loss values result in from loss functions which are used to know how much our system has accurately predicted and classified the given data, so finally now we now meet an important concept in the system's learning, the optimization function.

For each sample of data we get a loss or deviation from the expected outcome, all of these loss values are put together as an average of loss functions and conform the cost also referred to as the error function, and then it's time to direct this learning so the deviation is each time less and less every time through the change of these learnable parameters previously mentioned. Optimizers are algorithms that change all of these parameters and there are different types but we'll speak about the gradient descent one as is the most basic and most used optimization algorithm. Gradient descent consists in the search of the minimum within the function, so the parameters that minimize said cost function are to be found, this is achieved by looking at the first and second derivative points in the function as we are looking for the local minimums. As we can see in 2.5 in order to find these we take into account the slope of the mentioned first derivative of the function with respect to a value, and this slope will point to the nearest local minima. To picture this better we can imagine how we would be going from the peak of a mountain (high loss and cost), and as the neural network improves and learns we would be going to the deepest valley (low loss and cost) as seen in figure 2.6. In other scenarios we can find that there will be multiple parameters and so multiple slopes pointing to multiple existent minimums. There's also a variant of this algorithm called Stochastic Gradient Descent which is normally used in large dataset scenarios, as the explained gradient descent algorithm would have to compute the derivative of the function for all the data points, and this newly introduced stochastic one would pick randomly these data points at each step every time we need to calculate the derivative.

To summarize once we understand all the factors within an ANN and overall the objective wanted by the system, these learnable parameters previously mentioned are adjusted through an algorithm called backpropagation where we basically compute the learned weights and biases obtained in the previous step of the gradient descent algorithm. For each sample there's an adjustment that are used in an average, the resulting value is applied to the current layer and this process is carried out in each of the layers conforming the network, also it is important to state that this adjustment is done in by batches and each time all of the training samples go through backpropagation constitutes an epoch.

Finally after describing all the required concepts for understanding DL and ANN, in the following sections we will now explain ANN types and specially the ones that are important within the OMR field and the implementation of our project MureTools.

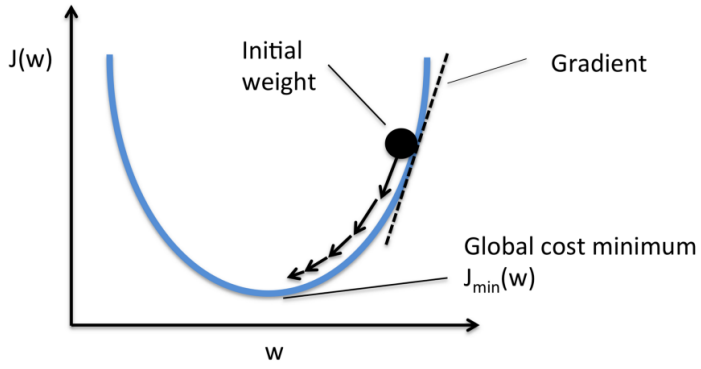



**Figure 2.5:** 2D Representation of the gradient descent where the slope of the first derivative is used to find the local minima (Source: S (2020))

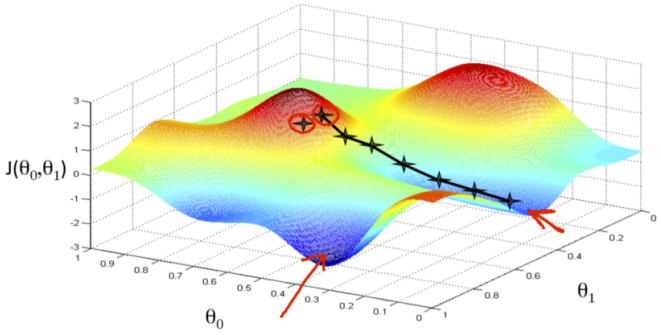



**Figure 2.6:** 3D Representation of the gradient descent where we can see the whole path followed from the first random value until the local minima achieved through multiple iterations (Source: Shin (2020))

### 2.2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks also shortened as CNN, are a type of neural networks used mainly in image recognition and processing, as it was designed for processing structured arrays of data such as images. At first it appeared as LeNET in 1989 using a CNN architecture and backpropagation, it was used for handwritten digit recognition, later on it regained attention in 2012 as AlexNet being one of the first CNN models implemented on GPUs and achieving a turning point in computer vision as it won ImageNet classification challenge, which consisted in classifying 1.2 million high-resolution images into 1000 different classes as it was expressed in Krizhevsky et al. (2012), and by huge margins which showed non-neural models to be almost obsolete. Due to this they became the standard for such tasks, and as an extension of this they are found within Optical Character Recognition (OCR), for human language processing or in our case, in MURETOOLS, for Optical Music Recognition (OMR) purposes.

CNN are known to be good at picking up patterns within these images inputted as arrays, and therefore that's why they are so useful for image analysis. But what is it that makes this ANN to have such a distinct trait? The reason underneath this resides in the hidden layers contained by this ANN, the convolutional layers.

Convolutional layers just like any other hidden layer, they receive an input and output a result that is fed onto the next layer, but the relevant part to us is what happens during the processing or transformation that is called convolution operation. To carry out such operations the perceptron makes use of a matrix called filter, actually there can be many of them, as these filters are the ones responsible for the pattern detection. To explain how this works let's remember that these filters are matrices, and that for each convolutional layer there will be a determined number of filters and finally each of them will be able to detect a specific pattern, let's see this with an example.

Imagine we have a CNN which is being fed handwritten digits like the ones in 2.7 and they are being classified into the respective number they are representing. All of these samples of numbers they have their own set of traits even the ones representing the same number, all of these traits will be detected by the filters and the convolution operation will take place, making use of the filter and and the input supplied in this case the previously mentioned handwritten digits. To showcase the convolution operation carried out we can look at figure 2.8 where we can see that each value from the pixels forming the number are mapped to a value from the 3x3 filter matrix computing the dot product, the dot product consists of multiplying the mapped corresponding values 1 to 1 and adding them together or expressed mathematically, the summation expressed in 2.3.

$$(a_1 * b_1) + (a_2 * b_2) + (a_3 * b_3)... + (a_n * b_n) \tag{2.3}$$

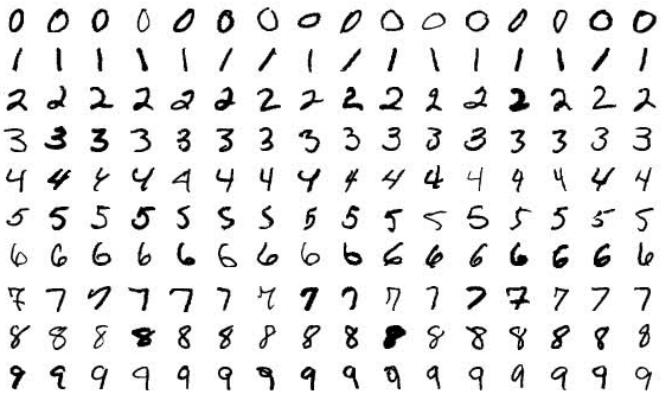


**Figure 2.7:** Extract from the MNIST (Modified National Institute Standards Technology) dataset, that is composed by 60000 small square 28x28 grayscale images of handwritten single digits between 0 and 9

The filters as previously mentioned can be identified as matrices, but to put an example of the different patterns that could be detected depending on the values used in its rows and columns, we can take a look at figure 2.9 where in this case we are looking for edges, from different orientations as the -1s corresponding to black would be the outside from the the shape given, the 1s corresponding to white would be the area looked for, the edge, and the 0s represented by grey would be the rest of our shape. The shape used will be a 7 from our handwritten digits.

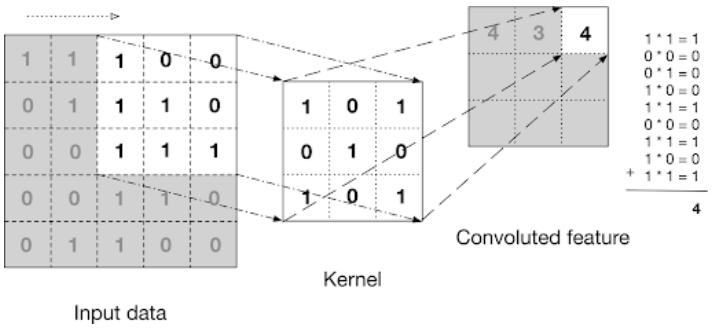



**Figure 2.8:** A dot product is carried out between the input matrix and the filter, resulting in a value stored in the output channel. Image extracted from "Deep Learning" by Adam Gibson, Josh Patterson

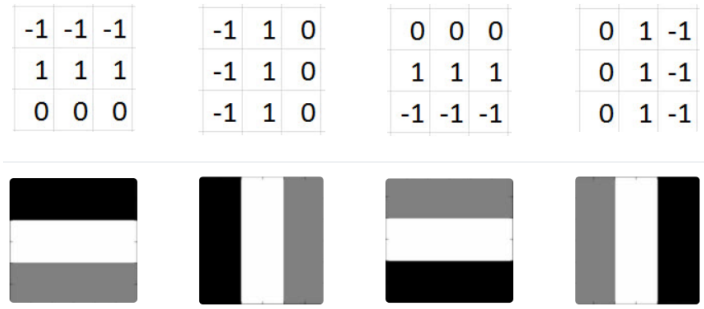


**Figure 2.9:** Convolutional filters extracting different traits from the image given, different patterns are looked in each one (Source: *Convolutional Neural Networks (CNNs) explained* (n.d.))

So to say from this product only one value will result and lastly this result obtained from the area will be stored in the output channel. Finally it is important to state that although at first our network would be using filters like these one and detecting only edges, as the process goes through the different convolutional layers and deeper in it, more complex filters will result that will detect more specific and intricate patterns. To illustrate the result extracted from these operations we can see the images obtained at figure 2.10 where we can see how each filter used results in a different trait extracted as the pattern looked for in each of them is different, in order from left to right: top horizontal, left vertical, bottom horizontal and right vertical edges respectively.

**Figure 2.10:** Output channels resulting from using each of the filters, the visible white pixels are the trait looked for in each case (Source: *Convolutional Neural Networks (CNNs) explained* (n.d.))

#### 2.2.1.1 Auto-encoders

This type of NN is a feedforward one based on unsupervised learning which is formed by an encoder, a decoder and an intermediate code which is only a single layer and will contain a compressed representation of the input originally fed to the encoder. The way it works is that, once the encoder applies compression to the input and the code is obtained, the decoder then will reconstruct through the compressed version the closest output possible to the original input. In order to visualize this more intuitively, figure 2.11 depicts how everything would look like.

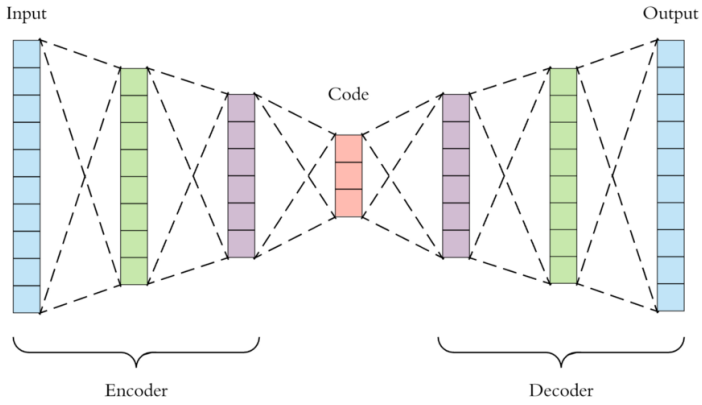



**Figure 2.11:** Architecture of an Auto-encoder where all the parts and feedforward NN can be appreciated (Source: Dertat (2017))

Although in the image showcased the decoder consists of a mirrored structure of the encoder, as it is expressed in (Dertat, 2017), it is not necessarily a requirement but it's typically the case, as the only requirement is for the size of the input and the output to be the same, as what is trying to be achieved is the same result in the input and the output.

In order to specify go deeper into its functioning it should be pointed out the parameters needed and that are influential for the training of this auto-encoder:

- Code size
- Number of layers of the encoder and decoder
- Number of neurons of those layers
- Loss function

There are multiple ways to organize this structure by adding or subtracting layers and also the number of neurons of these layers, which will be different to the single and different one in the code, as the code will be another component apart to the encoder and decoder with its own size, in addition, the smaller this code size is the more compression will be applied to the input. Finally to measure the resulting output obtained in comparison with the input, a loss function will be used.

It is important to state that there is another version of auto-encoders using recurrent neural networks (more specifically LSTM neural networks which will be explained later) that are used for sequence data, so without further due, this new type of NN will be introduced.

### 2.2.2 Recurrent Neural Networks (RNN)

In traditional neural networks there are cases when the concept of memory is needed within the network, here is where recurrent neural networks or RNN came into existence to solve this issue. Based off David Rumelhart's work in 1986 and seen before in Hopfield networks by John Hopfield in 1982, they are a type of neural network designed to learn sequential patterns or in other words patterns that vary through time, like for example number series or a sentence. Thanks to this internal memory we are able to take into account all of the data fed until the current step.

This memory is achieved by feeding into the perceptron its own output as input. To explain this we have to understand the difference between RNN and the traditional feed-forward neural networks, as noticeable in figure 2.12. In feed-forward networks the information only moves in one direction from the input layer, through the hidden layers and finally to the output later, this means that the current situation is not considered, there's no notion of order in time, each time it goes to the next layer the information from the past step is forgotten and not taken into account. On the other side RNN considers the current input as well as what it learned from the inputs received in previous steps. This is why RNN are ideal for text and speech analysis, so for example let's say we give to our feed-forward neural network a word like "learn", as layers would occur all of the achieved information would be lost and would not be possible to make sense of the other characters to predict the expected character, in other words the resulting characters from previous steps (for example "lear") would not influence the decision for the current step to predict "n" and so it would be harder for our neural network to achieve the given word "learn".

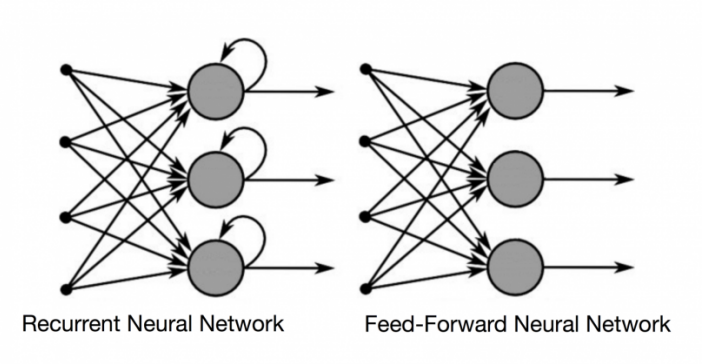



**Figure 2.12:** RNN have additional information of the current state within the perceptron as opposed to the feed-forward neural networks. Image extracted from Niklas Donges' article on RNN

These loops as found in figure 2.13 cause the perceptrons to feed themselves the output as input and so tackles the problem of sequential data, however two more problems surged from

this: the exploding and the vanishing of gradients.

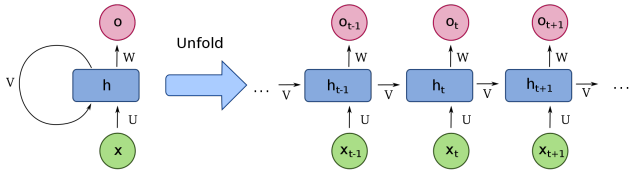



**Figure 2.13:** Structure of a basic recurrent neural network. Image from user fdeloche via Commons Wikimedia

As explained previously at the start of this section, the gradient determines how much the learnable weights change with regard to the change in error, or differently expressed, the slope of the function indicating the direction where we are supposed to learn quicker, our local minima. So what would happen if the gradient assigns values too large to the weights, due to the multiplications carried out in the backpropagation, the network then would continue to grow and grow without really finding the most optimal way of learning, though in order to easily deal with this we would be able to do so by just truncating these gradients with huge values. But in the other hand if the gradients were too small as a result the weights will be changed indeed in a slow way with what we could call as low as ”vanishing” values, in other words what we could expect from having a slope (gradient) close to 0 and furthermore the network would take stop or take too long to learn, analogically to solve this problem the Long short-term memory (LSTM) networks appeared.

#### 2.2.2.1 LSTM

LSTM are networks based in RNN that differ from the latest ones by extending their memory, so that they can remember information for long periods of time, so the data persists during far more frames than what a regular RNN would be able to persist. This memory is achieved through a gated cell that determines whether that information is relevant enough to be stored or not, this judgment is carried out through the weights, that are adjusted as the network learns, in other words as time passes it will learn what information in relevant and what is not.

There are three types of these mentioned gated cells: input, forget and output gate. These gates are presented as sigmoids ($\sigma$), which are activation functions that decide if a value passes or not, so they output a number between 0 and 1, being respectively completely deleting and keeping the pertinent value. As what we can see in figure 2.14 through these 3 types of gates we can introduce new input (input gate), affect the current neuron with the resulting output (output gate) or not store it at all (forget gate). It is important to state that these gates are just neural networks with weights and biases, and through them the flow of information is regulated within the sequence chain.

To summarize it is important to state that RNN are indeed really important in OMR as the nature of the data seen in this field is in the form of sequences, having in mind that many music scores are written in staves and recognized as sequences, and that with the help of

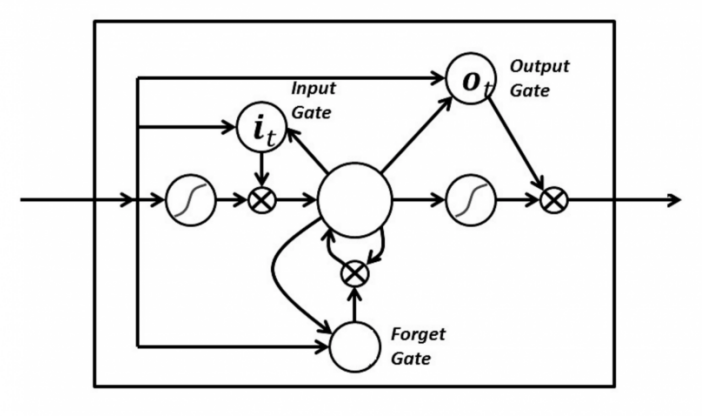



**Figure 2.14:** Gates found in a basic LSTM memory architecture, single cell. Image extracted from Niklas Donges' article on RNN

LSTM we are able to keep these music pieces in context.

#### 2.2.2.2 GRU

There is also a different alternative to LSTM, those are Gated Recurrent Unit (GRU). Gated recurrent units, although being an advanced cell putting context and memory into RNN just like LSTM, its difference lies in its number of gates as it possesses less than a LSTM and thus it has less complex structure, also this is the reason why in terms of model training speed GRU is faster than LSTM.

GRU has two gated cells instead of the three being used by LSTM. In this case the two being used are the update gate, that as its name expresses, will decide whether the cell state is going to be updated or not with the current activation value, and the remaining gate will be the reset one for deciding how much of the past information is forgotten. Just like LSTM, in this case the gates can also be noticed as sigmoids ($\sigma$) activation functions, everything is visually represented in 2.15 where it is appreciated that the workflow followed is far simpler of the one in LSTM.

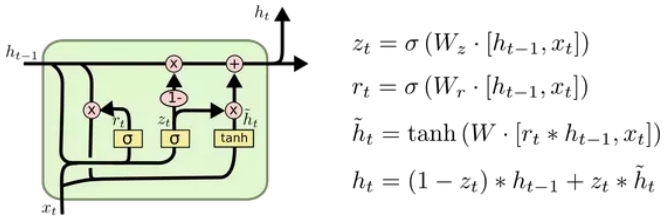



**Figure 2.15:** Gates found in a GRU architecture with the operations of the update gate ($Z_t$), reset gate ($R_t$) and the hidden states ($h_t$) (Source: Rathor (2018))

In the end both options were conceived to solve the vanishing gradient problem of a standard RNN, but there are cases where one of the two options is preferred, specifically in terms of disposing of a larger dataset, a LSTM is preferred as it should remember longer sequences

and thus get better results than GRU, and on the other hand GRU are simpler, easier to modify and also faster to train with, so in conclusion depending on the context one option will fit better than the other and vice versa.

So now finally we will also put in context all of these so needed RNN in OMR, with models proven in this field, like the end-to-end and sequence-to-sequence. As well as the concepts found within them that will be explained.

#### 2.2.2.3 Connectionist Temporal Classification (CTC)

These neural networks will be used in our models implemented, like the End to End one, but all of these processes will be more thoroughly explained in the Design chapter 5, as it will make more sense on context with the operations being carried out on our endpoints using this. These two models are commonly used in fields where data is sequential, like speech recognition, text recognition and many other fields where a sequence is processed as an input, retaining its state while processing the next sequence of inputs. To put it another way, in these scenarios unlike in traditional feed-forward networks where inputs are assumed independently and actually taken into account, in these sequence data scenarios, each input is dependent on the previous one.

In the first case, for the prediction of sequences through Connectionist Temporal Classification (CTC), it comes from the combination of CNN and RNN, to take from each layer its advantages and strong points within the field we are approaching, in this case OMR, to recognize the elements we find in music scores. In the end we want to pass all of these images as sequences and label them, and so for that we are going to need CNN for extracting the desired feature sequence, then RNN will propagate information through this sequence extracted, predicting each frame extracted.

And at the same time along with this Convolutional Recurrent Neural Network (CRNN), a CTC loss function is considered, this operation will be used for focusing on getting alignment between the sequences of unsegmented input data, like for example aligning words to an audio signal, but many other fields show this kind of scenario, for example handwriting recognition and more specifically a handwritten music score.

To explain CTC more precisely, it calculates the loss between the continuous unsegmented time series and a target sequence by summing over the probability of all possible alignments of input and the target label, producing a loss value which is distinguished with respect to each input.

So in order to achieve this we got a matrix with probabilities of all the labels plus one, that will be the ”blank” or separation one, and all of these will be summed over for each time-step. The process that the CTC would follow is the following:

1. Assign the highest probability
2. Merge repeated labels
3. Remove the ”blank” label

In the end it is not needed to have already segmented training data (as this is not how it is found in the real world and is a really burdensome task) and also there is no need for post processing of the operation's output. To picture this better in 2.16, there is a graph comparing CTC and a framewise classification that would label each time-step or frame of the input provided.

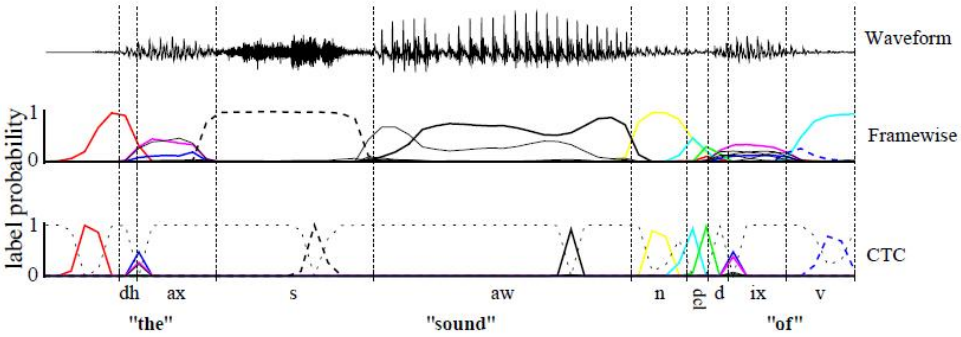



**Figure 2.16:** Comparison between a framewise and CTC approach for predicting phonemes in a speech signal, the lines are the output activations corresponding to the probabilities of that phoneme at that time, the separations are ”blanks” that will be removed later on (Source: Graves et al. (n.d.))

So in other words through CTC it is possible to be able to train with only the unsegmented input, without needing to worry about the separation between these sequences, and also after this CTC operation is carried out, it is only necessary to decode, to collapse and remove the blank labels that uses to solve the problem of label repetition in time-steps.

#### 2.2.2.4 Sequence to Sequence (Seq2Seq)

In this last case of RNN based models, Sequence to Sequence is just a model that takes a sequence as input and also outputs one thus its name.

This model is basically compound of an encoder and a decoder, which are just stacks of RNN, although they could use one of the other options mentioned throughout this whole section, so they employ use LSTM or GRU, as this task is sequence based and what is desired in these scenarios is to capture the context of the sequence and preserve it through the whole process. In order to do this, vectors of hidden states accumulated are used, so a hidden state vector is passed to the next RNN cell and so on, until it reaches the end of the encoder. Once that happens, the final result of this encoder is called embedding, and just like the previous hidden state vectors passed they can be of any size but in most cases they are taken as a power of 2 (as in DL, training is often carried out on the GPU and using power of 2 allows

for the GPU to take advantage of optimizations, result of its processing design) proportional to the complexity of the complete original sequence.

Lastly the decoder is fed with this embedding and through that encapsulated context from the encoder, it predicts successively, using the previous hidden state to achieve the next prediction and so on.

However through this whole process when input sequences are too long it is difficult to keep the context, that is why another mechanism called Attention was introduced.

Attention just like the cognitive one, is a mechanism used to ”focus” in specific hidden states from the vector each time as it predicts, focusing on what is more important for defining the context. This is achieved through ”attention” weights that indicate which one is the next most important one for the prediction. In order to obtain these attention weights, a ”context” vector is built each time step, by a weighted sum of all the input hidden state vectors, later on during the decoder processing these hidden state vectors will take into account the new weights generated by the last context vector and so on. The process is carried out as follows:

1. Obtain context vector
2. Concatenation into hidden state vector
3. Calculate new attention weights (attention vector)

In 2.17 the steps expressed above are visually presented so the elements involved in each operation are clarified.

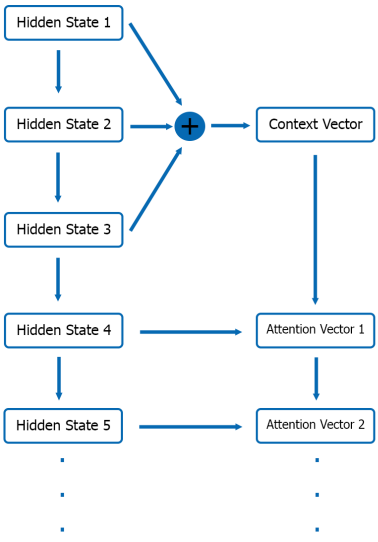



**Figure 2.17:** Scheme describing the whole process carried through for obtaining the attention weights

Ultimately to recap, the final model showcasing the entire process is shown at 2.18.

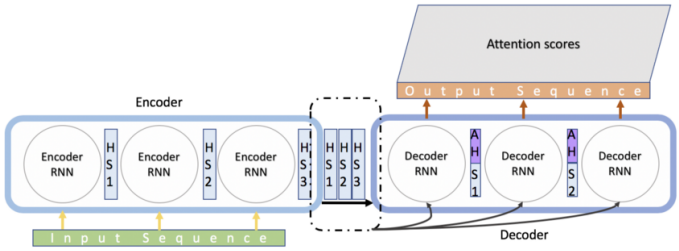



**Figure 2.18:** Scheme showing the whole resulting model with Attention. HS stands for the Hidden State vectors and AHS stands for the Attention vectors that also take into account the hidden states thus the HS (Source: Dugar (2019))

## 2.3 Introduction to Microservices Architecture

Finally to end this chapter it is important to also bring in the architecture that will be used in this project.

Microservices Architecture as the name states, consists of group of small services, running independently so through lightweight mechanisms they provide these services by being deployed autonomously. Through this way of working the application is set up as group of loosely coupled, collaborating services.

By not being tied up under the same structure, there are some advantages to highlight:

**Increased scalability and flexibility:** Due to being able to add new services for new needs more easily and using the most fitting languages and technologies.

**Improved productivity and speed:** When developing as these services are smaller, smaller teams can pick them up, causing greater agility, communication and thus productivity.

**Easier to maintain and build:** Managing the code becomes easier as it is divided in services and so the testing and deployment is easier.

**Fault tracking and isolation:** In the case of a failure in the application, it is easier to track down to a service, and despite becoming unavailable it won't interrupt the normal functioning of the application as long as proper handling exists for the failures.

The cause behind the above advantages reside in really related concepts. As modularity of these services sparks the previously expressed strengths.

When it comes to the communication of these services, it is possible through synchronous protocols such as HTTP/REST or asynchronous protocols like Advanced Message Queuing Protocol (AMQP), in the case of this project both will be used, not only for consuming the endpoints but also for having our own message queue manager for all the tasks and processing that is going to be requested by the user. As in a common microservice scenario, having

a message queue manager allows for accepting multiple requests of these microservices and having the possibility of giving these tasks out evenly, for multiple providers of the requested microservice to process them efficiently.

To have a depiction of how an architecture like this would look like, we can take a look at 2.19, it is also important to highlight that Microservices Architecture has become in recent years more and more popular.

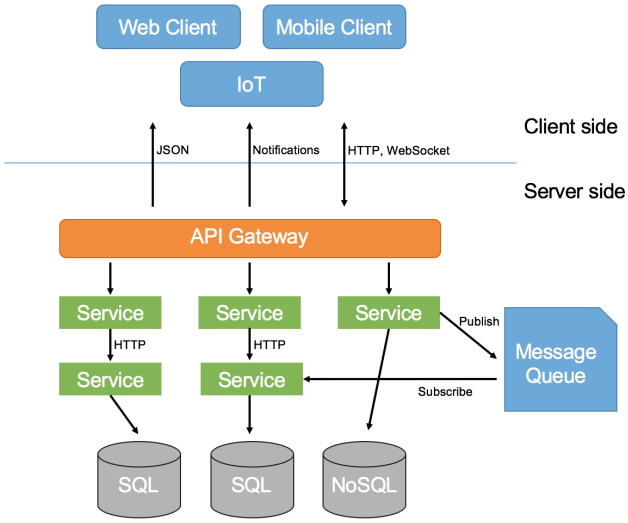



**Figure 2.19:** Example of a Microservice Architexture which also employs a message queue manager (Source: Dinh (n.d.))

# 3 Objectives

Now in this new chapter the objectives to be accomplished in this project will be defined. MureTools's objective can be summarized as a supporting system for Optical Music Recognition (OMR), in a way that allows training and evaluation tasks in this field to be achieved more easily with a systematic, automatized and more comfortable approach than the ones used until now.

Also it is important to state that this supporting system role will take place within the framework and context of an already existing platform from the University of Alicante called Music Recognition Encoding Transcription (MuRET), which is a machine-learning based research tool that allows for different processing approaches to be used and produces both the expected transcribed contents in standard encodings and data for the study of the transcription process, so historical music archives can be transcribed and digitized to a digital structured format like XML-based ones to be conserved (Inesta et al., 2019).

First and foremost MureTools's objective will be establishing an API (Application Programming Interface) that allows the user to train the desired model with a given set of data that will correspond said model's format, and return the results of the training, evaluation and the actions taking place.

For the execution of the above objective there will be some concrete objectives to accomplish during the implementation of this project:

- Understand and study the processes involved in extraction of data and labelling within these documents in this field of study (OMR).
- Design an API able to carry out automatically and concatenating tasks as recognition, labelling and automated machine learning.
- Implement an usable and accessible interface that allows users interaction for every task wanted to execute.
- Evaluate the results obtained from the tasks carried out in a scientific and established standard way being intelligible for everyone in this field, so it can be shown to every user through an usable and accessible interface.
- Compare the results obtained from the models and contrast, so the best model with its components can be saved and be uploaded to MuRET's server.
- Design a system capable to manage processes, queuing and status notifications to meet the requirements of MuRET's specifications as well as the standard development of internet services.

Explaining thoroughly the first main goal already expressed, as a result of all of these accomplished objectives what is expected to have is a system that allows tasks originally executed in OMR, including training, labelling and evaluation within a model and dataset selected by the user. All of these choices will be available to be made in a quickly, easy and comfortable way carrying out all the needed tasks at once.

In the end other objectives also achieved as a consequence of accomplishing the above explained targets would be the following ones:

- Make the use of Neural Networks (NN) and evaluation of the results, for investigational and research purposes, easier to get into and understand for newcomers to this research field as vast as Machine Learning (ML).
- Create an environment where multiple NNs can be used in the same way, as NN have many intersected needs than can be supplied through the same ways.
- Create the possibility for the system to escalate and provide the previous explained services to other different NNs with different configurations within the models and so on.

Throughout the entire development of this project, these will be the objectives to fulfill. In the next chapter the system's functional requirements will be analyzed and all the specific tasks required to meet these requirements as the development takes place will be defined in detail always bearing in mind the first and main goal of MURETOOLS.

# 4 Analysis and Specification

The definition of how the objectives were proposed for this project MureTools depend on this section where all the system's analysis, requirements and specifications will be described throughout this chapter. Thanks to all of these there will be an option to turn to when following the entire project's design and development.

All of the previous will be established following IEEE830 standard (IEEE, 1998), where there are stated good practices for the definition of analysis and specification of software requirements. As the standard is not fixed in format and can be adapted to our own convenience, only the essential parts to describe what is relevant to MureTools's development will be commented.

Before starting to define the analysis and specification it is important to state that this project exists within the context of a larger platform, MuRET, a project carried through and maintained by the University of Alicante.

## 4.1 User profiles

First of all, the kinds of users existing in the application will be explained, so we know the skills and roles they are going to carry out within MureTools, as well as a basic description that summarizes them.

As we can see from above's table 4.1 there are 3 recognizable user profiles: administrator, researcher and the user. Administrator and researcher profiles have more intersected and related features, as the main big difference is that the administrator takes a more web development related role and the researcher just the OMR's investigation one which either way matches with the administrator. Meanwhile the regular user is the one who is starting in this ML and OMR field and wants to approach it in a more understandable and easy way.

## 4.2 Restrictions

Through MureTools's development there will be restrictions to bear in mind in conjunction with the requirements, so that everything when designing and planning the application turns out to satisfy every one of these needs. As it is seen in table 4.2 all of these restrictions are from a determined nature and will have a direct consequence on our project.

## 4.3 Requirements

Lastly we will describe the mentioned requirements that altogether with the above restrictions shape the making of our application. In this section we can be able to distinguish two

**Table 4.1:** User profiles perceived in the application

| User type | Features | |
|---|---|---|
| Administrator | Description | Owner of the system<br>with all kinds of permits and rights<br>to the data and applications services |
| | Skills | High technical and educational level.<br>User with enough knowledge in distributed systems<br>and web software development<br>as well as machine learning and OMR to some degree |
| | Role | Capable of accessing<br>the monitoring of all services in the application,<br>controlling all the users activity and<br>solve all the architectural and network<br>issues to happen within its use.<br>Also they implement all the<br>next features to add to the application. |
| Researcher | Description | Users interested in the development<br>of ML systems focused on OMR.<br>They are mainly driven by investigation in the field. |
| | Skills | High academical level and wide knowledge about ML and OMR systems.<br>The range of software and web technologies<br>in which they perform their activities<br>is really wide to define them accurately. |
| | Role | Will focus in the development<br>and use of OMR technologies as well as just ML<br>in general applied to the previous said field of study. |
| Regular user | Description | Anyone who is interested in OMR technologies and ML.<br>This kind of user might be attracted to use it<br>so they can approach these study fields<br>in a more easy and comfortable way. |
| | Skills | Must have basic computer knowledge,<br>can use a web browser on different devices.<br>They understand OMR or ML related technology to some degree. |
| | Role | They will use the application to interaction with OMR or ML<br>in a more engaging and easy way<br>to understand or explain related concepts. |

**Table 4.2:** Restrictions table

| Identifier | Type | Title and description |
|---|---|---|
| HW01 | Hardware | **Servers Limitation**<br>Our application MureTools will be employed within the servers supplied by the Language and IT Systems Department from the University of Alicante. The limitation of these servers tough is not exactly known, chances are probably that there will be a limitation on the amount of tasks they can carry out. |
| HW02 | Hardware | **GPU capability**<br>When it comes to activities as training, predictions and others alike<br>we will need some certain GPU capability. We will need to manage our available resources in a proper way. |
| US01 | User | **Knowledge on the field**<br>One of the things that distinguishes our application from others is the fact that for using it there are many concepts that could be difficult to explain in an easy way for the users starting on this field of study<br>and one of the objectives we want to accomplish with this project is to allow the usage to this kind of users too. |

types of requirements, depending on if they have a direct relation with a functionality the infrastructure must accomplish due to the user's demands (functional) or they have indirect relation (non-functional), as these demands can be carried out without them.

Now in the next subsections and tables we will refer to functional requirements and non-functional requirements as two different entities and proceed to explain them.

### 4.3.1 Functional requirements

Here the requirements which have a direct impact on the system as they define the basic behaviour when responding to input from the user. They describe literally what the system must do and if they are not present the system won't work properly as it is intended. In the table 4.3 we will describe all of them as well as define their nature.

### 4.3.2 Non-functional requirements

Finally in this subsection we will present the requirements that do not define tasks the system must do but rather how these tasks should be carried out. So opposite to the functional ones, these are not necessary for the system to work as a whole, and be able to be used for its original purpose. In the next table 4.4 we will see how all of our previous defined demands should be executed.

**Table 4.3:** Functional requirements table

| Identifier | Profile | Description |
|---|---|---|
| FRADMIN01 | Administrator | The administrator can check the status of all the available services as each of the individual features. |
| FRADMIN02 | Administrator | The administrator can access any of the data saved within the application. |
| FRADMIN03 | Administrator | The administrator can add and test every new feature registered. |
| FRADMIN04 | Administrator | The administrator can grant and revoke permissions to researchers as well as any other regular user. |
| FRRES01 | Researcher | The researcher will be able to define and add new features and configurations of the tasks to execute related to the training and definition of models. |
| FRRUS01 | Regular user | The user can select the model to train as well as the matching corpus that is required to train with. |
| FRRUS02 | Regular user | The user will be able to see the training logs matching the tasks taking place underneath the system. |
| FRRUS03 | Regular user | The user will be able to save the wanted model freely or mark for the system to compare and save the model with the best results achieved. |
| FRRUS04 | Regular user | The user will be able to save the logs and results obtained from the trained model. |
| FRSYS01 | System | The system will notify the user if there is an incompatibility as a result of any of the selected models, corpus or offered options in the training. |
| FRSYS02 | System | The system will manage the resources optimally for the tasks demanded as well as queue all of these tasks and notify the user the successive executions. |

**Table 4.4:** Non-functional requirements table

| Identifier | Profile | Description |
|---|---|---|
| NFRSYS01 | System | The system will use a resource manager for deciding when to perform a task. |
| NFRSYS02 | System | The system will recognize states so the user is able to be notified by them. |
| NFRSYS03 | System | The application will receive the appropriate and matching configuration and data from the user and if not there will be possibility to change the affected factors to the ones wanted by the system. |
| NFRSYS04 | System | The system will always be available through a fixed web domain, that allows every user with internet connection to access and use it freely. |
| NFRSYS05 | System | This project's code will be widely commented and under a version code system so every developer can contribute to the project to create new features and fork at will. |
| NFRSYS06 | System | All the project's collected data will be under an appropriate open-source license, so the application and the information within it remains public and free for everyone. |
| NFRSYS07 | System | The user interface will be responsive, meaning that it will adapt to any device is being used on. This way the screen will adapt according to if a smartphone, a tablet or computer is being used. |
| NFRRUS01 | Regular user | The user will be able to see every available feature to use as well as the results achieved by the different tasks carried out in the application. |

# 5 Design

In this design section we will define many of the solutions to the previous stated functional requirements. This way we will have set a guideline for MureTools's development to follow.

In this project we will have 3 distinguished parts that will be explained in the next sections. First we will define them and explain their relations between each other.

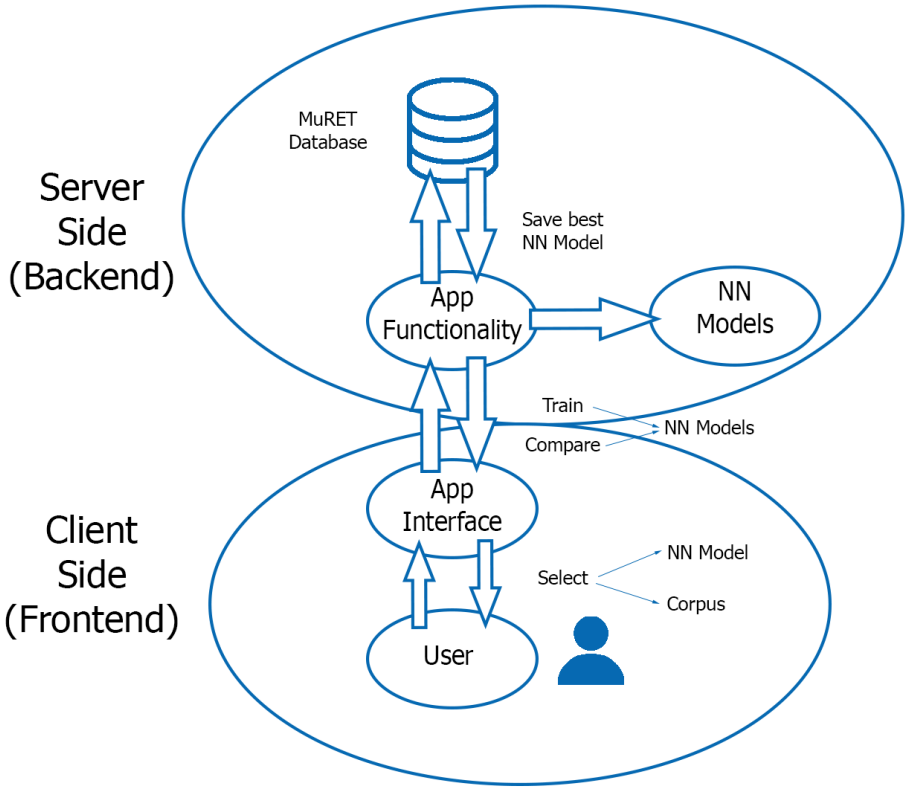



**Figure 5.1:** Scheme of the involved parts in MureTools as well as their relations.

As we are able to see in the scheme 5.1 we have a backend that contains all the neural networks models available to train with. So these models are ready to be fed on data and other related configurations, all of this input will be introduced through a frontend or interface which the user will interact with.

So to summarize these are the bare bones to MureTools and now we will proceed to explain them more deeply as well as all of their relations with each other. As seen in 5.1 we will begin from the bottom with the user, and proceed until we reach the end of our scheme, explaining all the encountered tasks and actions involved.

## 5.1 Frontend

Firstly the interface design of our application will be really minimal and straightforward, so the user can easily recognize and understand how to request the wanted NN model to train, as well as parameters related, and the corpus used for it.

The most desired objectives in this frontend was to have a usable and accessible interface, and as the required functionalities within it were not too complex, it was intended to be achieved without a framework, so only through pure and plain HTML through Jinja2 the template engine used by FastAPI (which will be introduced later in the chapter 6 Methodology), JavaScript with some jQuery, and CSS also with Bootstrap. So in order to later implement it, the whole design was sketched through mockups as can be seen in 5.8, and advanced slowly around it.

### 5.1.1 Colors and typography

When it comes to the colors used, the palette was partially inspired in the blue tones by the ones used in MuRET, and along other tones like the green one were in fact included to transmit trust, security, correctness and accuracy among other desired values, which want to be related to the steadily functioning and accurately provided services and metrics within the application. Also there were some colors used in the background form and the task cards, used to relate every task executed with the model that was trained in each case, all of these colors are showcased in 5.2.

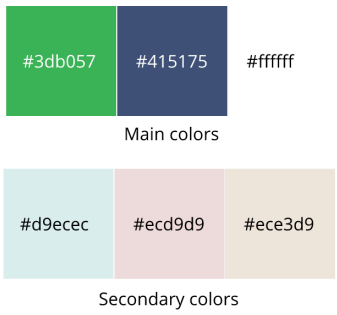



**Figure 5.2:** Color palette from MureTools, the main colors are the one used throughout the whole web application while the secondary ones are used to color code the models

On the other hand, talking about the typography used for MureTools, Open Sans was selected as the one due to being a clean and modern sans-serif typeface thus being specially designed for legibility across print, web and mobile interfaces. In addition it is also available via an open source license which makes it free to use for personal and commercial purposes. This typography is shown at 5.3.

Due to not having a standalone version of MureTools for smaller devices like mobiles and tablets due to time constraints as expressed in chapter 8 Conclusions and Future Work, at least these devices were taken into account through this typography and the responsiveness

achieved through Bootstrap utilities as showcased in 5.4.

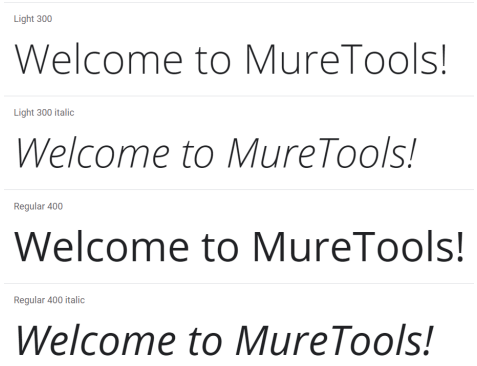



**Figure 5.3:** Open Sans typography sample

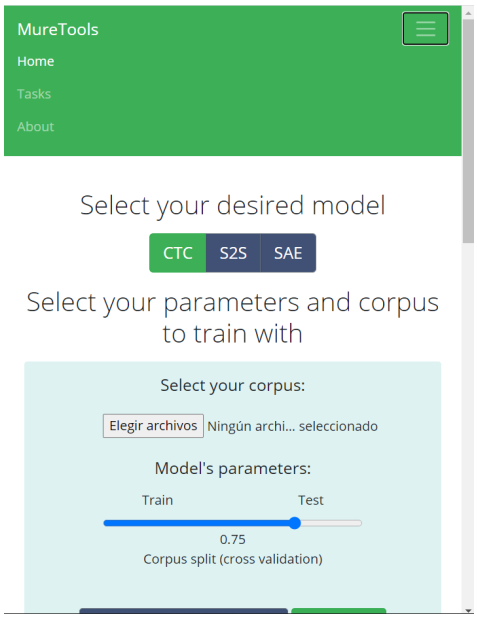



**Figure 5.4:** MURETOOLS adapting to a smaller size

### 5.1.2 Progress, different versions and mockups

To begin with, it was required to be able to allow the user to introduce specific parameters for each model, that is why everything was conceived within the same page through one unique form. Due to this, there had to be a variable section for all the parameters that would change as soon as the user would select the model desired to train with. If there were missing parameters or incorrectly introduced, then an error message would appear over the corpus and parameters section as seen in 5.5.

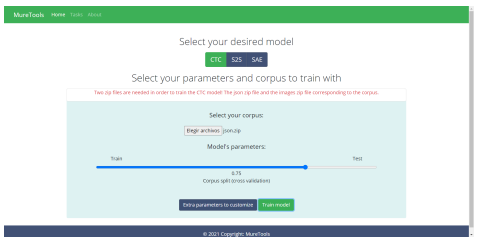



**Figure 5.5:** Message returned as a result of the validation of the data sent

Apart from this, once the tasks are requested what was needed in order to keep track of these, was a place for them to appear with their related information so they could be distinguished and noticed by the user. At first it was planned to only have them underneath the form, as it was only a space for them to be shown, but as later on it was thought of the possibility of scaling on more displaying features and more related sections, another section was created for displaying and filtering, as it is appreciated at 5.6.

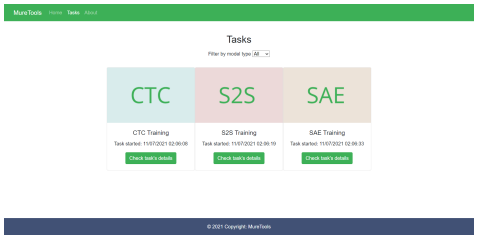



**Figure 5.6:** Tasks page where all of them will appear with the option of also filtering

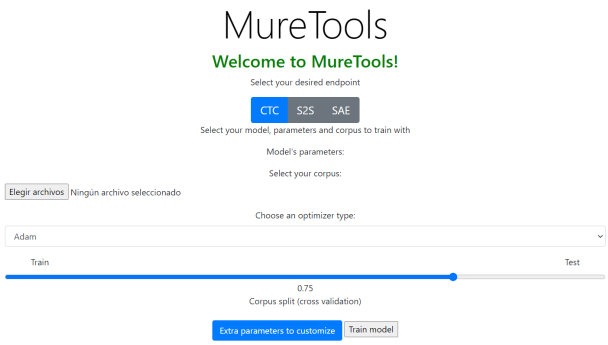
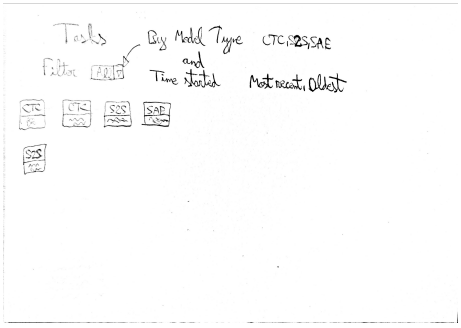
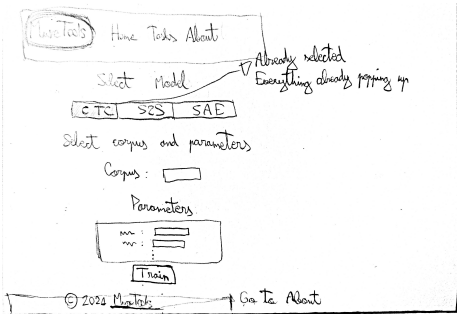


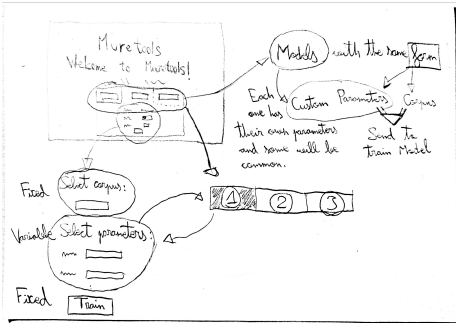

**Figure 5.7:** First version of the MURETOOLS interface developed off the mockups




**(a)** Tasks page




**(b)** Home page




**(c)** First drafts

**Figure 5.8:** Mockups made for MURETOOLS design

## 5.2 Neural networks models

Now we will explain in detail all of the different neural networks available to train, as well as the process followed in each of their respective endpoints. We can distinguish 3 of them: End-to-End, Musical Encoder and Document Analysis. All of these will be explained thoroughly in the below subsections.

### 5.2.1 End-to-End

In this endpoint there will be different steps to follow. So first, after the data is loaded (images and agnostic sequences, that is an encoding representing the output of the music symbol recognition) then the vocabulary is created, where sequences provided from the input dataset will constitute a dictionary for the predictions to take place within the system, these sequences are nothing more than character, which will be given a unique integer, for each one of them.

Then the creation of the CTC model takes place, which we can configure with different parameters as the input shape that the model is supposed to expect for later on starting the training, and the size of the recently created vocabulary. This way we can provide the input size for the first 2 dimensional Convolutional layer and the vocabulary size is used as the unit number for the Dense layer later on. So that finally the returned results are the training and prediction models. Which will be distinguished due to the fact that the training one will

perceive the mentioned CTC and the prediction one not, which later on will be used just for storing this CTC.

From the interface as stated, all the different parameters relative to the creation of the CTC model and the configuration will be passed. This includes the train and test percentage of data splitting, which means how much of the total dataset is used to train with and how much is used test the resulting model, as through this method the model is tested with data that has never been fed so it is totally new for the model, which serves as a good test. This is normally used in order to avoid the phenomenon known as overfitting, which causes the model to "get used" to the dataset provided thus not generalizing for performing properly with other potential datasets that we might use.

Once the training has been carried out, the model will be evaluated through a metric called Sequence Error Rate (SER) that will be checked each epoch, so that if the SER in the current epoch is better than the one from the previous one, it will be stored as the best one thus the model will be saved as the best one until that moment, as it is required for later to upload the model with the best results to MuRET.

It is important to also highlight that the metric used, the SER, expresses the ratio of incorrectly predicted sequences with at least one error, so it only takes into account the perfectly predicted sequences, as stated in Calvo-Zaragoza & Rizo (2018), which makes it a more reliable comparison than other metrics computing the average number of operations in a sequence to match other which would be unfair in the case of agnostic and semantic sequences as they are different in length due to their ways of encoding.

Also in this End-to-End solution it should be pointed out that when planning the use of this method and designing the methods for extracting data to train with, in order to have good results is necessary to dispose of big amounts of data to feed in order to obtain good results, and not only that, but they also should be labeled so the ground truth is imperative.

About the resulting metrics it's important to state that the Sequence Error Rate (SER) will be shown to the user, and use resources like line graphs among others to represent all of this data efficiently. Also there are other metrics like the Character Error Rate (CER), which is a widely used metric in speech recognition systems and therefore in end-to-end models, that refers to the percentage of characters that were incorrectly predicted just as analogously Word Error Rate (WER) metric would be the same to the percentage of words. But we should remark the fact that in this case, in OMR, there's no standard metric established as of the moment.

Referring to the CRNN-CTC it's convenient to explain some concepts surrounding what is happening in the processing of this endpoint.

In the task of recognizing written text, more specifically in our case written musical notation, the NN used are normally consisting of convolutional layers (CNN) to extract a sequence of features from the corpus given, then later on recurrent layers (RNN) are used to propagate information through this sequence, resulting in character-scores for each sequence-element which gives us a matrix with scores that indicate which character is more likely to be in each sequence.

With this matrix there are two important tasks to be carried out and both of them have

a common intersection point, the CTC operation. Basically what they share in common is that they are going to be achieved by this operation.

Before defining and describing the CTC operation, it's important to explain why this was chosen. As this CTC operation achieves to avoid the need to annotate a prepared data-set on a character-level, as well as the necessity of processing the final result for getting the final text from just the character-scores returned.

The way CTC works is the CTC loss function is fed with the output of the NN and the corresponding Ground Truth (GT). Then every possibility of the GT is tried, so the score of a GT text is high if the sum over the alignment-scores has a high value.

So in the end, CTC encodes the text given also with the help of a blank pseudo-character representing a separator that is used so that there aren't duplicate characters, although this will be ignored in the decoding, so they will be removed.

Later on the loss function calculation takes place, feeding to that function the training samples (image-GT text), to train the NN:

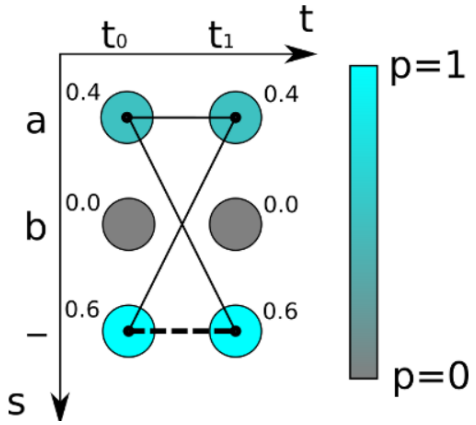



**Figure 5.9:** Output matrix of the NN. The character-probability is color-coded and is also printed next to each matrix entry. Thin lines are paths representing the "a" character, while the thick dashed line is the only path representing the blank "" character (Source: Scheidl (2021))

Taking as an example the one provided in Scheidl (2021) and taking a look at the above matrix 5.9 we can see that the loss is calculated by adding up all the values of all possible alignments of the GT text given. So following the expressed case, the total results would be:

- "aa" – $0.4 \times 0.4 = 0.16$
- "a-" – $0.4 \times 0.6 = 0.24$
- "-a" – $0.6 \times 0.4 = 0.24$
- "-" – $0.6 \times 0.6 = 0.36$

Assuming the GT as "a" or as "" (blank) we have to try only with paths within a length of 2 due to the fact that the matrix has 2 time-steps (x axis), altogether the results for both possibilities, taking the previous calculations into consideration, would be the following:

- "a" – $0.4 \times 0.4 + 0.4 \times 0.6 + 0.6 \times 0.4 = 0.64$
- "" – $0.6 \times 0.6 = 0.36$

Now we want the NN to be trained so that it outputs a high probability (ideally, a value of 1). This is why we want to maximize the product of probabilities of correct classifications and at the same time minimize the loss of the training dataset (being this loss the negative sum of log-probabilities), the loss value of a single value will be just the logarithm of the computed probability and a minus in front of it. During the training of the NN, the gradient of the loss with respect to the NN parameters (e.g. weights of convolutional kernels) is computed and used to update the parameters.

Finally the last stage of this CTC process would be the decoding, where we want to calculate the most likely text so in order to accomplish this we will look at the output matrix of the NN. We will make use of the best path decoding algorithm which consists basically in two steps:

1. Take the most likely character per time-step
2. Decode by removing duplicate characters and blanks, the remaining represents the recognized text

So in conclusion, the best path is taken and all the duplicates and blanks resulting from the encoding and removed so we can finally get the final recognized text from this whole CTC process. Through this decoding algorithm we are able to easily make an approximation of what the text might be as we see in the example from 5.10. Although as this method achieves an approximation there could be scenarios where the resulting text is wrong comparing it to the GT text provided, for example if we used this same decoding on the first displayed matrix, the most likely text would be "", although as we proved previously with the sum of probabilities the "a" would be in fact the most likely text to be recognized.

### 5.2.2 Musical Encoder

In order to build this model, a NN known as Sequence to Sequence or more commonly Seq2Seq will be used. There are three noticeable elements that will make it possible: an encoder, a decoder and an attention block. The way this is going to be built, the sequence consisting of agnostic and **kern files, will be given to feed the model getting in the first place into the encoder, that is nothing but a stack of recurrent units, then through this they will get propagated, so that each cell of LSTM (GRU is also possible) accepts an element from the sequence and propagates it forward. The result produced by the encoder is a vector that encapsulates all the internal states memorized from the encoding process, and that will serve as context for the decoder to make predictions. The outputs of the encoder are discarded, only these states are relevant for the resulting vector, so they will continue to influence the decoder that will use the context of these sequences in order to predict the correct resulting sequence.

To start, the input that will be given to feed as mentioned previously will consist of agnostic and **kern formatted files, which means that for every image or labelled part of the document

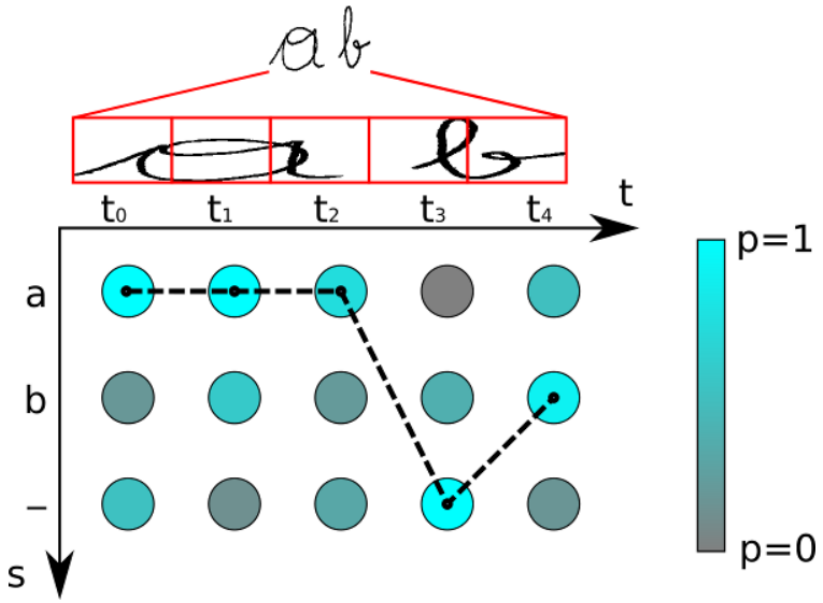



**Figure 5.10:** Output matrix of the NN. The thick dashed line represents the best path, corresponding to the first step enumerated about the process (Source: Scheidl (2021))

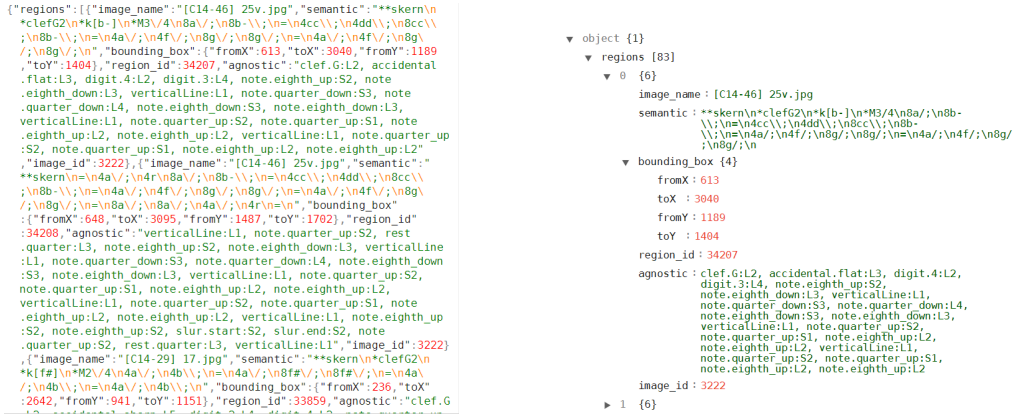
we are using to train, there will be a **kern sequence with important information relating to the semantic part of the sequence, as well as one with the agnostic one. Also the information related to the respective image of that sequence of notes, this includes the bounding box and ID given to identify that image as can be seen in 5.11.

Furthermore something notable to remark in this model is the use of a mechanism called Attention, through which the resulting vector or embedding outputted by the encoder possesses a weighted combination of all the input states and that will influence each decoder output in each step. This means that in order to make predictions thanks to the weights, the input state given with more weight will be taken into account first for the prediction and so on through the whole decoding phase.

Additionally this type of model that adopts this Attention mechanism is also known as Transformer.

When putting together this model, the parameters selected to let the user introduce them within it, were the number of recurrent units or RNN neurons employed, size of the embedding, this representing the vector outputted by the encoder and that is intermediary between encoder and decoder and the number of K-folds for the cross validation, that serves as the number of groups the data gives is divided into, so there is an evaluation carried out by using unseen data for the model as it was not used during the training. So the steps to employ this method are:

1. Shuffle the dataset randomly
2. Split the dataset into a number K of groups
3. Then for each group:




**(a)** Original JSON file loaded **(b)** Beautified for readability

**Figure 5.11:** A fold of the dataset provided to feed the Seq2Seq model in MURETOOLS, where the agnostic and **kern format is appreciated

- Use the selected one as testing dataset
- Use the remaining ones as training dataset
- Fit the model using the training set and evaluate it with the testing one
- Store the evaluation score, discard the model and repeat again with the next group

4. Calculate the average of the stored scores. This will be the model's performance metric.

Finally, after the training has been carried through, analogically to the End-to-End model, the evaluation, model saving and subsequent request for MURET to be uploaded, will be the same, also using the previously introduced SER metric.

### 5.2.3 Document Analysis

First it's important to state that in OMR all the musical notation from images is read with the objective of automatically exporting the content to a structured format. Due to this computational process being as complex as it is, this task is usually divided into different stages, the first one is the Document Analysis, which in the context of the model implemented in this project, will take an approach making use of selectional auto-encoders (SAE).

In this stage we will distinguish all the traits and information given from the different sources of information so we can categorize in the possible elements within a musical score: background, staff line, musical note or lyrics(text).

So in the context of MURETOOLS, in this stage with the necessary data we will be able to see our respective metrics from the endpoint. This data will be a zip with images so we can feed the NN models and also related to these images we will receive a JavaScript Object Notation (JSON) file with all the different regions to their respective images.

For the sake of this process there will be a set of auto-encoders one for each of the wanted regions to be categorized in. This is the advantage and difference between this method and a traditional pixel-wise classification approach. So these SAE will be four in total as can be

observed in 5.12.

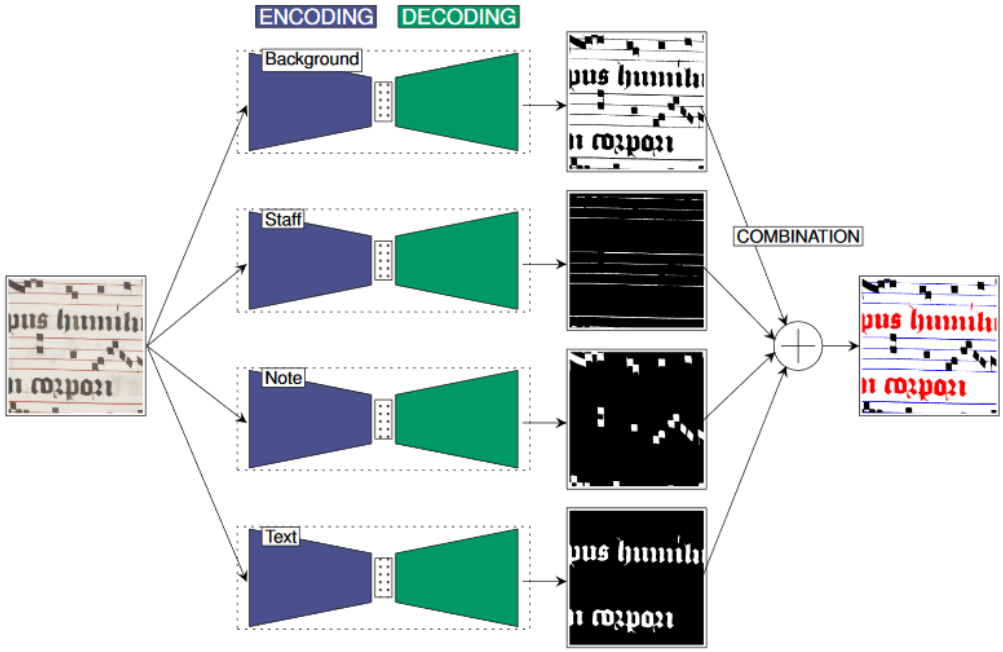



**Figure 5.12:** Graphical scheme of the SAE-based1-vs-all approach for document analysis of music scores images. The outputs of the individual SAE are represented as grayscale masks in which the white color represents the maximum selectional value. Coloring for the final combination: background in white, music symbols in black, staff lines in blue, and text in (Source: Castellanos et al. (2018))

The result of these auto-encoders are later on combined to obtain a global analysis of the document.

Now with all of these needed factors, our system will train the NN model with the given information, returning to the user resulting metrics such as precision and Intersection over Union (IoU). These two metrics will allow us to understand the accuracy on the dataset provided.

For the IoU there are two factors we need: ground truth and the prediction from our model. Due to this metric we will be able to tell apart the different bounding boxes when exists overlap between them. If the prediction is completely correct the IoU will be equal to 1, and the lower the IoU's value the worse the prediction result is. So with one square being the ground truth and the other one the model's prediction, IoU could be determined as seen in 5.13.

So in the end this approach making use of these SAE, will have the advantage compared to a traditional Convolutional Neural Network (CNN) in the context of analysis and recognition of music score documents, that this way we will have the possibility to reduce considerably the time required to train the model to predict the category wanted, and also some features that will give us more flexibility on the process workflow. This is achieved by basically two

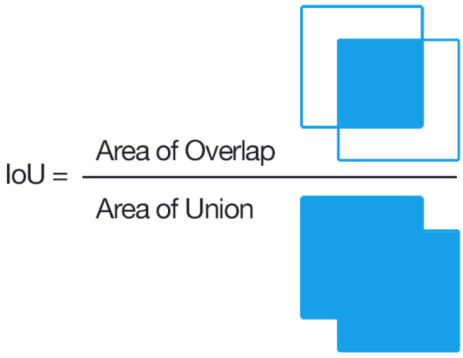



**Figure 5.13:** Computing the Intersection over Union is as simple as dividing the area of overlap between the bounding boxes by the area of union (Source: *Intersection over Union (IoU) for object detection* (2016))

facts that this new approach offer:

- No need to prepare the training set as only the ground-truth of the targeted category of the SAE is required (staff, lyrics, etc.)
- Each prediction provided by each SAE can be processed separately so we can apply different thresholds and configurations to resolve inconsistencies

In this model as a difference to point out with respect to the previously introduced ones, although it follows the same pattern for evaluation and model saving and upload into MuRET as the previous ones, there is a new relevant metric in this case that was not in the others, this is the F-score or F-measure. This metric is used to test a model's accuracy on a dataset through creating a relation between precision and recall, as it is defined as the harmonic mean (type of average used for numbers representing a rate or ratio) of these. Recall just expresses the correctly classified examples (true positives) and its misclassified ones (false negatives), precision on the other side relates to the true positives and the ones misclassified as positives (false positives).

## 5.3 Backend

Finally how all of these endpoints are treated as well as the request and the possibility of a process manager that queues all of the tasks to be carried out, along with state notifications which will be able to serve as ”traffic lights” for determining the next task to execute.

### 5.3.1 Microservices Application Architecture

As MureTools has a really specific objective, and specialized tasks that are provided specifically for a larger purpose, focusing only in the functionalities needed, it was decided to be built under a Microservices Application Architecture so that also this application could become easier to scale and faster to develop, opening the possibility to change and adapt to new needs when training and just consuming the endpoints in general by the larger platform MuRET which is being provided with these functionalities.

**Table 5.1:** Table comparing the most important differences between Microservices and Monolithic architectures

| Microservices | Monolithic |
|---|---|
| One specific goal | Entire environment for all goals |
| Modularity. Easy to track errors and failures | Everything built in the same environment, difficult to track errors and failures |
| Easy to scale and develop | Difficult to add new features for new needs |
| Lightweight | Heavy and large, long building and deployment times |

A Microservices Application Architecture is recognized by some characteristics that distinguish it from the traditional Monolithic Architecture, as seen in the above table 5.1. From all of the given points, it is obvious that the most fitting architecture for MureTools would be the Microservices one.

### 5.3.2 Queuing system

Finally in the backend, it was critical to implement a queuing system, creating the possibility of asynchronicity within the application between tasks, as these tasks would be all long enough to require this. Through this feature the user could create multiple tasks and continue using the application by seeing the data and plots within different tasks, as others would continue training and evaluating in the background.

The in-built Background Tasks provided by FastAPI (originally from Starlette, as FastAPI is based on Starlette) were conceived for simple operations that needed to happen after a request like email notifications or simple data processing, so asynchronously performing heavier background computation as it was MureTools's case, sparked the need for a bigger tool like Celery. This technology would allow for more flexibility as background tasks could be ran in multiple processes, and also important for the long run and future scalability, in multiple servers.

Through Celery's way of working the pipeline would look like the scheme seen at 5.14, where it is perceived that a message queue manager (broker) will give the tasks accumulated in the queues to the workers to dispatch, so they are being attended and completed with an update of the result in Celery's result backend, that the user will be able to retrieve at any moment.

Along with Celery there are a couple of in-built result backends which use AMQP, this is the protocol used by RabbitMQ in order to produce messages for the consumers (workers) to pick them up and process them, so by using the same protocol it is possible to send the results to the client application. The result backends using this are AMQP and RPC backends, which can be used instead of an external option like for example Redis. AMQP is considered now deprecated and also the employed one, RPC, is a good option for scenarios where the process that initiates the task is always the process to retrieve the result, which is more fitting and scalable in the case of a single user than the one offered by the

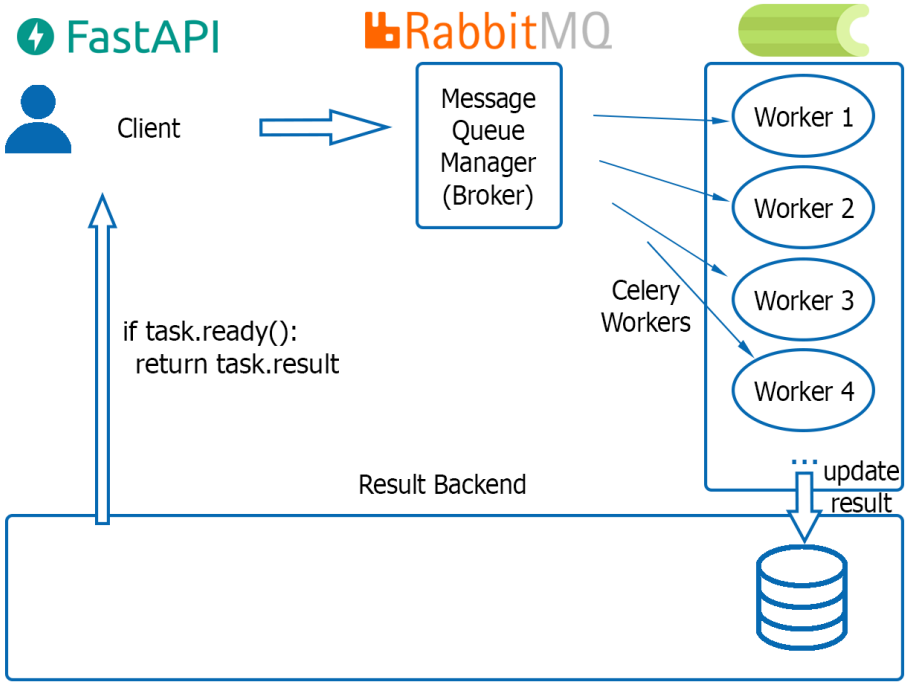



**Figure 5.14:** Scheme of the pipeline following how a task is created and dispatched. As it is appreciated RabbitMQ was employed for the broker role

AMQP backend. The reason for this is that the AMQP backend uses a whole results queue per task call, meanwhile the RPC one uses a result queue per client thus its name Remote Procedure Call, although it limits in the case of having another entity wanting its results, but in this one the user that produced the task will be the only one to also consume this result.

Finally this results backend will contain all the results updated once the task is finished, with all the metrics and logs accumulated through the training and evaluation of the model, and that will be retrieved in order to update the user with the tasks' status and data passively, and also to allow the user download this data actively.

In addition, this design supports concurrency, so to say, it is possible to multi-process by performing concurrent execution of tasks, which is relevant in the long run for future scalability and needs that may appear.

# 6 Methodology

In this chapter it will be explained how the project was approached in terms of planning and organization, software and technologies selected as well as the whole division of the work carried out in order to achieve the proposed objectives. In order to do this, all the stages that took a part in this project will be introduced along with the software and technologies that were necessary to accomplish said objectives.

Additionally to showcase the quantity of work done in each of the stages, captures of the monitored time in Toggl will be shown, so it is appreciable the importance and priority of the tasks in each stage, as well as being able to keep an eye on how the development is going and keep a steady and regular workflow as far as possible. Toggl is a time tracking tool where all the time inverted can be registered and divided in fields or tasks, as shown in 6.1. Also the division used for the tasks was in 4 big types: Machine Learning (ML) for all the machine learning and deep learning related tasks of implementing the models through TensorFlow and Keras, Web Development (WD) for all the tasks related to developing the web application including frontend, backend and the queuing system, Miscellaneous (MISC) for all the tasks related to investigating and researching concepts and technologies that had to be implemented and used respectively, among other things, and finally Memory (MEMO) for documenting this whole project as well as its related tasks like the creation of schemes, figures and tables among other things. These tasks are visually color coded as seen in the following figure 6.2.

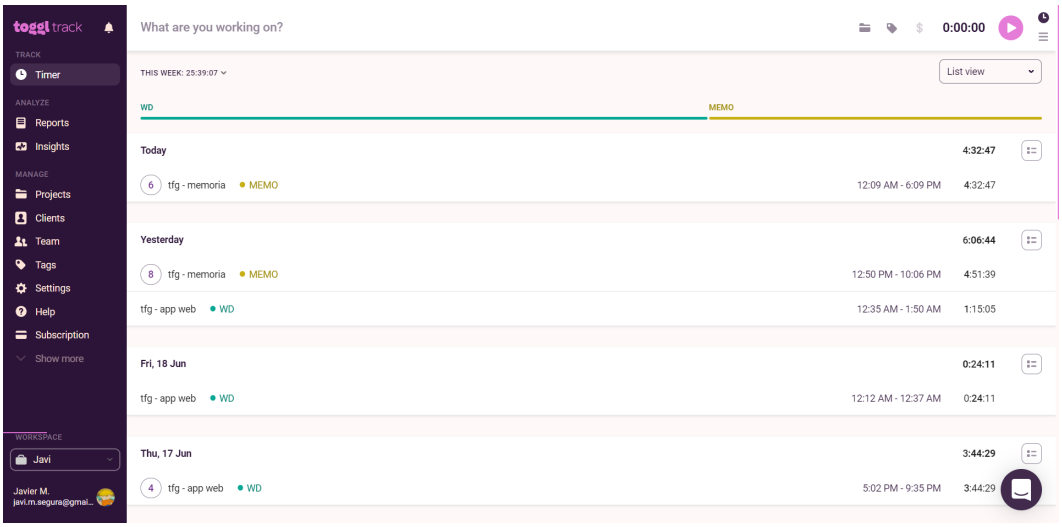



**Figure 6.1:** Capture of Toggl, software used for tracking the time spent in each task in a project


**Figure 6.2:** Grouping of the tasks in MureTools, blue is for the ML and DL related tasks, green for the web development ones, red for miscellaneous and and yellow for documenting the whole project

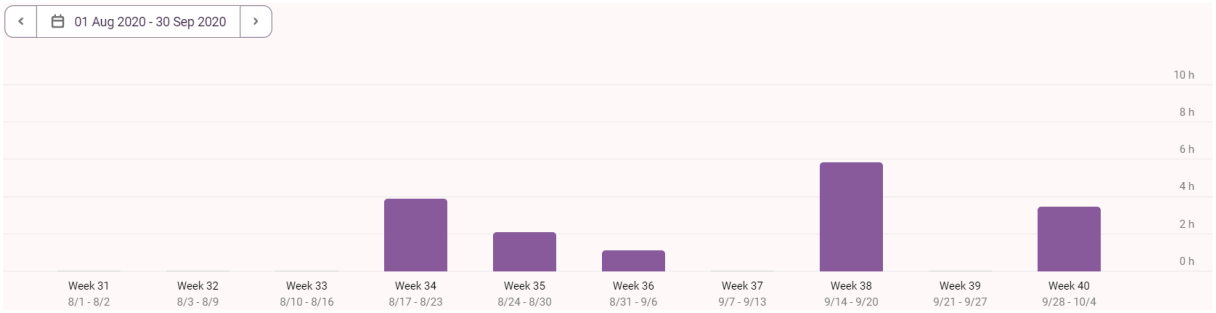
## 6.1 Stage 0: Introduction to Machine Learning

Before ending the last academic year in 2020 it was desired to decide the topic of this project, as it was planned to start slowly researching and getting a grip on the related disciplines. After finally deciding the topic of this project, the first steps in this journey were connected to getting used to ML and all its related concepts as I had no previous background in anything from this specific field. And also it was necessary not only to understand the concepts within this field, but also the software and the technologies used normally as an standard that is why it was pursued to have a first contact in a practical way so the concepts could also settle down more easily. This stage took place on and off through August and until part of September as can be seen in 6.3.

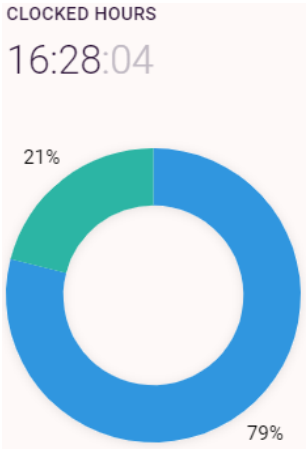
### 6.1.1 Python, Jupyter Notebook, Anaconda, Tensorflow and Keras

In order to have this first practical contact and after learning the basic theoretical concepts, multiples resources were researched, and finally ended up picking up a YouTube tutorial series where exercises where episodically carried out building around what was done on the previous episode and introducing new concepts each time. This was a great start to Python, a programming language mainly used in the data science world and thus also ML, getting to know its way to use and all the types of data structures that would be used like lists dictionaries or sets among others. In order to run this code an environment called Jupyter Notebook which also would allow for a place to include all the libraries used in deep learning: Tensorflow and Keras, as seen in 6.4. These two libraries provide the functionality needed to build models with the desired layers and parameters, as well as being able to train and evaluate and many more features. Furthermore to install all of this and have a virtual space where different versions of these two libraries as well as other that may be needed to treat the data so we could have different environments with different libraries and versions, Anaconda was used and along with that conda, numpy and other libraries that would be needed.

So with everything set up correctly, the first models and neural networks were built, and the first exercise consisting of a CNN classifying images of dogs and cats, as well as other experiments were carried out.




**(a)** Hours per week in Stage 0




**(b)** Hours division in Stage 0

**Figure 6.3:** Total hours worked in Stage 0

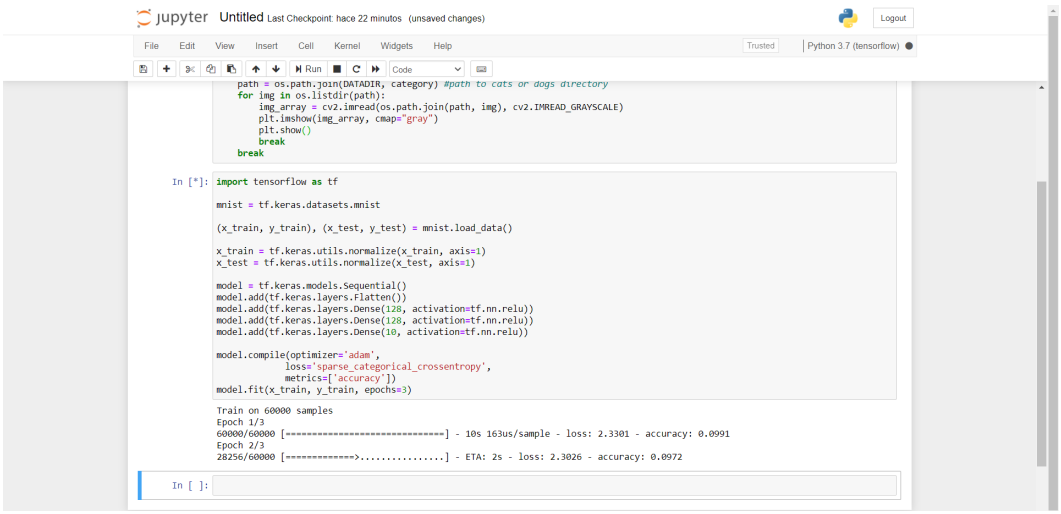
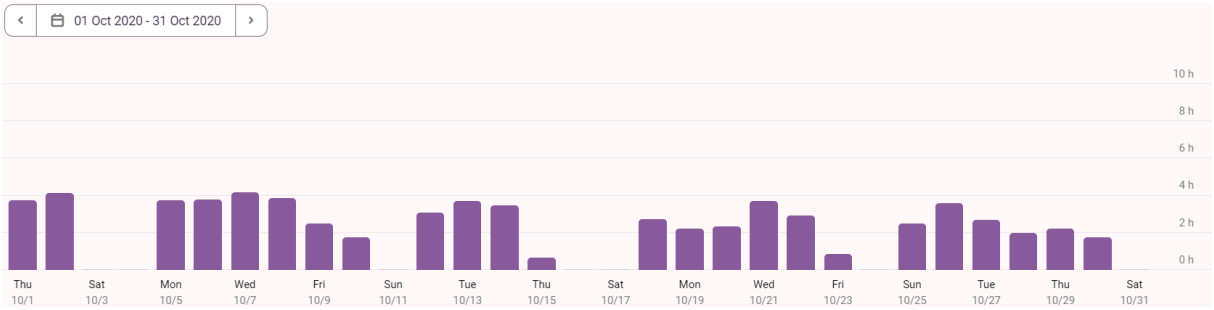


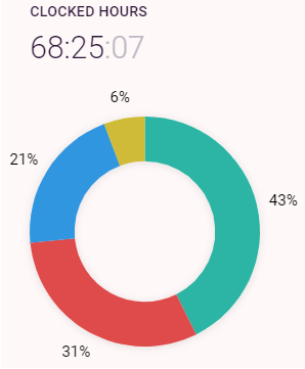

**Figure 6.4:** Capture of a notebook in Jupyter Notebook where it is possible to run Python code along with different libraries, and specify the Python environment desired in each notebook

## 6.2 Stage 1: Application bare-bones and first requests

Once the academic year officially started and after having our firsts contacts with this project's topics, it was time to actually narrow down to MureTools's specific scenario and how it was going to be tackled down. As it was going to be necessary to implement not only these NN, but also a web application with its functioning frontend and backend, so that all of these NN could be hosted. In this stage, the next proposed objective was to create the foundations of the application and the bare-bones of the frontend and backend to get the development started and reiterate around this adding new features. This stage took place during October as expressed in 6.5.




**(a)** Hours per week in Stage 1




**(b)** Hours division in Stage 1

**Figure 6.5:** Total hours worked in Stage 1

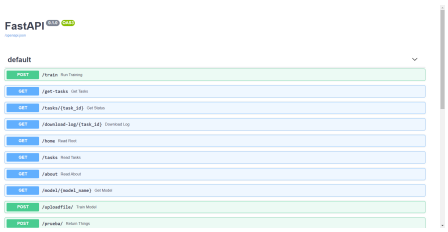
### 6.2.1 FastAPI and Postman

The technologies added during this stage, with the aim of building a simple Application Programming Interface (API) with a simple frontend were FastAPI and Postman.

Although at first, before getting into this stage it was mentioned Flask and advised, as it was a really minimalist and lightweight framework to build through Python that was already used by my tutor, that was going to be the selected option when this stage came but after talking again about this subject, another framework was advised with really similar characteristics but it was more recent and an up to date documentation. Finally this was the

selected framework to develop with.

FastAPI allowed for a really quick API building, and so the first endpoints were created, paying attention to the way these were declared, the types of responses and parameters, and also the creation of documentation on the fly as it was built, as appreciated in 6.6. Also how files were uploaded as well as other parameters through forms was investigated, as although I had some experience making requests, there were some things to learn about how certain things were received and converted in the backend, and in which way, Body, FormData, Headers among other things.




**(a)** Different endpoints with their methods

**(b)** Parameters and response from the /train endpoint

**Figure 6.6:** Captures of the documentation created by FastAPI

So although at the end of this stage finally a simple frontend was used to test, at first and through most of this stage a tool called Postman was employed to create the requests and test, as this is normally used to consume APIs, so the endpoints and sending of data were tested through this, as can be appreciated in 6.7.

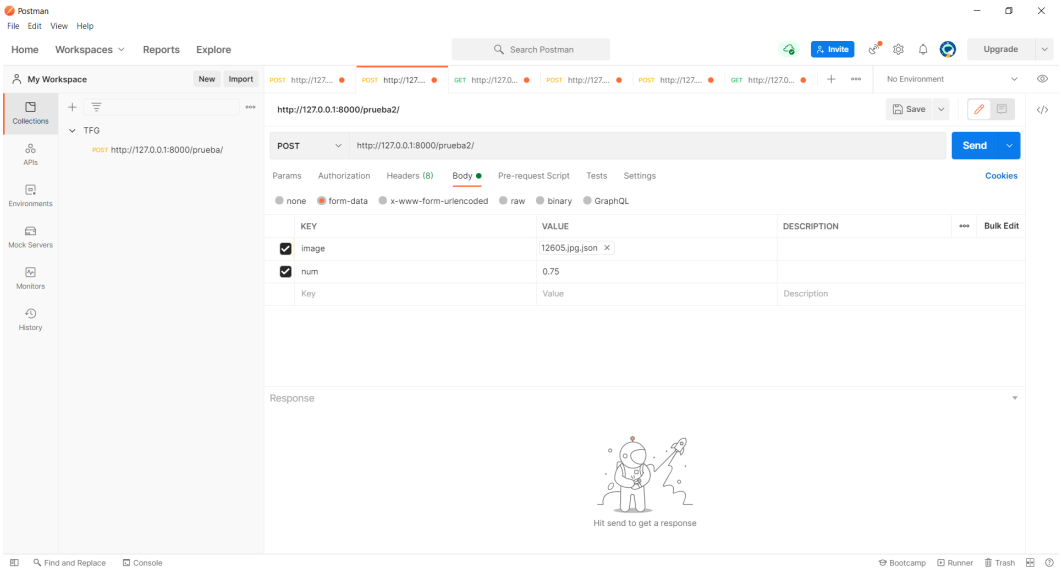
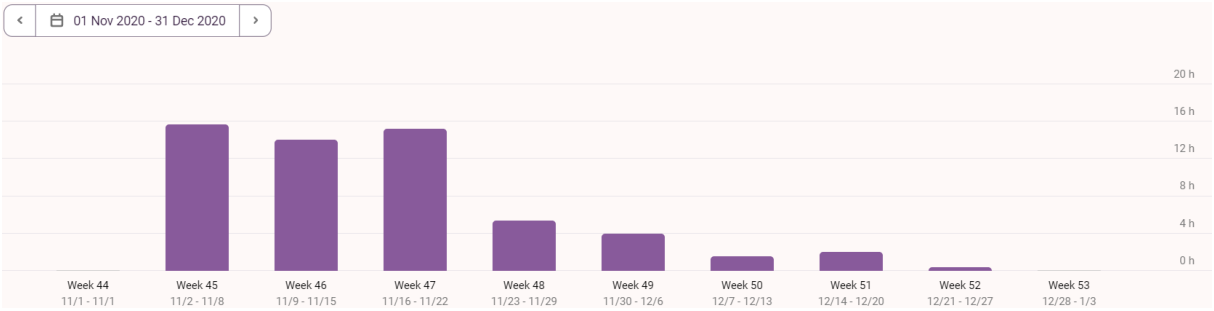


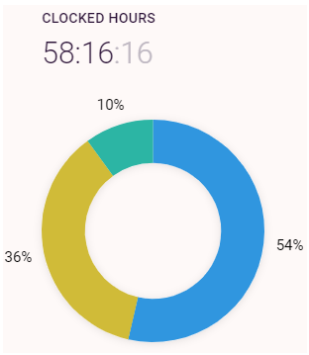

**Figure 6.7:** Capture of Postman where requests were done to the API, specially at the start of the project

## 6.3 Stage 2: First Model End to End CTC

Finally the first model to implement was introduced, as each time I had to implement a new model from the relevant ones mentioned for the OMR tasks, a meet was held with my tutor in order to explain roughly the concepts and pipeline of this one, thus for the other ones similar meets would be held likewise. This stage took place occurred during November and part of December until Christmas holidays, as the work during the vacations was irregularly carried through on and off, and not considered relevant and with a fixed objective to accomplish as can be perceived in the chart at 6.8.




**(a)** Hours per week in Stage 2




**(b)** Hours division in Stage 2

**Figure 6.8:** Total hours worked in Stage 2

### 6.3.1 End to End CTC

In this stage although new software was not added per se, there were many concepts to be learned not only about DL within OMR and sequence data oriented models, but also for training and testing this model it was needed to get used to the dataset provided to use as the corpus, that consisted of images of the music scores and the respective agnostic formatted data within JSON files, that constituted the ground truth, as well as the loading of this data into Python variables to feed the model. The dataset used for this model was a sacred manuscript dated from the second half of the XVII century, stored in Pilar de Zaragoza's Cathedral.

In addition for testing, some parameters were changed already through MureTools's frontend parameters and used in the model, like the split proportion of data destined for training and testing respectively.

## 6.4 Stage 3: Metrics, Files and Second Model Sequence to Sequence

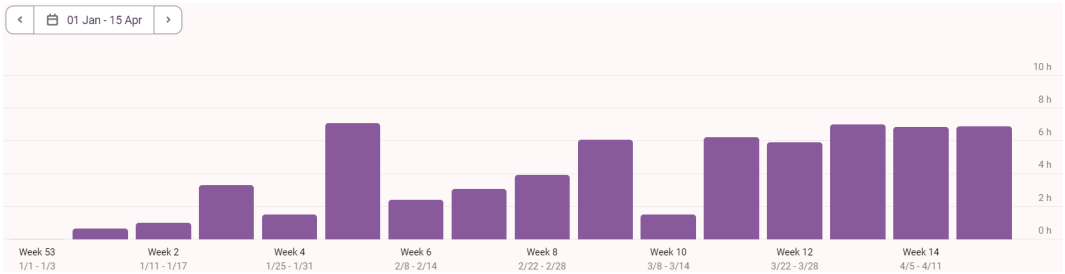
At this point as it was achieved a fully functional model which went through all the processes correctly, it was time to start getting all the metrics that would be needed to display the data on real time, and as a summary after the whole task was finished, for the user to download. This is the reason why it was required to also get into file writing and directory creation, as well as providing them the correct way to feed the charts, frontend, and as a download endpoint compressing them into a zip file.

Apart from all of this, the new model Sequence to Sequence was introduced which meant new concepts and processes to learn, on top of that the queuing was also supposed to take place in this stage. As it is appreciated from December onwards the remaining stages were longer in time as also there were more open matters to take care of, this one ranged from January until the start of April approximately. Despite the great amount of time all the objectives were not accomplished which would slow down the overall development of the project and create the need for more time and objectives in the next and final stage. This was also one of the reasons why some of the features mentioned in the Conclusions and Future Work chapter 8 had to be scrapped out from the practical implementation. All the work carried through this stage can be seen in 6.9.

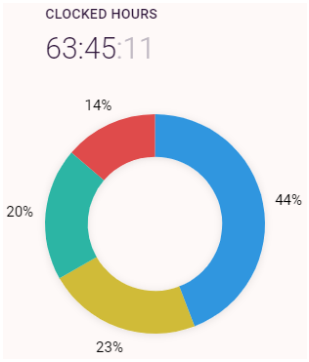
### 6.4.1 Callbacks, GPU usage and Queuing

So for achieving the mentioned metrics, a feature from Keras called Callbacks where used, which provided a way of defining the behaviour in the model's training when certains event happened, like the end of a batch or an epoch among among others, being able to write the metrics during the training and also storing all of them at the end.

Finally, here in this stage it was supposed to take place the investigation and development of the queuing system for the application, as well as looking into more detailed ways of employing the usage of the GPU and memory for said queuing system, sadly due to complications related to some of the implementations in this stage, like for example with the second model, the project was slowed down quite a bit a resulted in having to delay this system for the next and final forth stage, as well as hindering the practical implementation of the integration of GPU usage and memory in said system.




**(a)** Hours per week in Stage 3



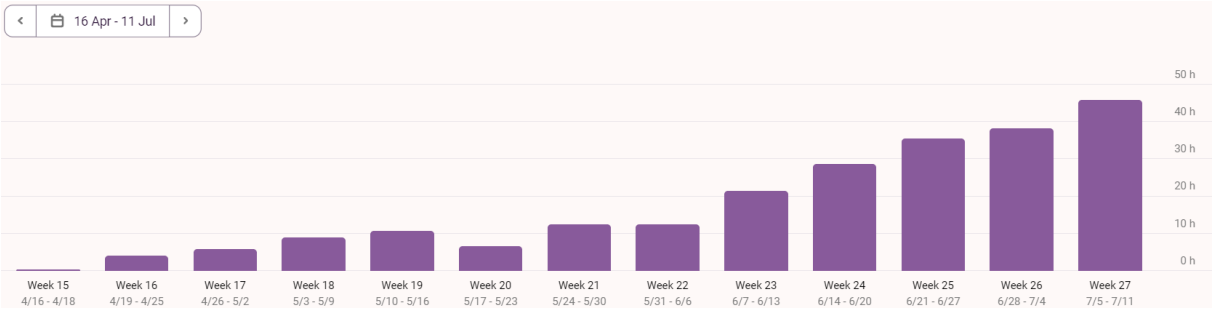

**(b)** Hours division in Stage 3

**Figure 6.9:** Total hours worked in Stage 3

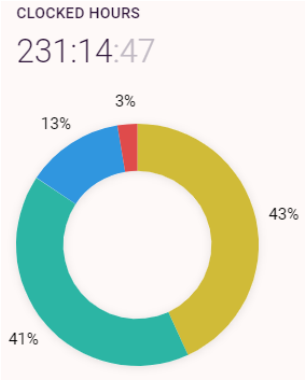
## 6.5 Stage 4: Third Model SAE and finishing the Minimum Viable Product

Lastly, as there was still one model to be introduced and multiple features to be included, within the last stage it was prioritized to finish the minimum viable product. This not only meant to complete the last vital features needed, but also to connect all the features achieved until that point into a proper system with a functioning workflow. Since some of the described features were working as standalone functionalities and not fully connected between each other.

This is the reason why this was prioritized and finishing the minimum viable product meant consummating the queuing system which was started on the previous stage and supposed to take place then, as well as implementing the last model and finishing the last details when it came to plotting and final debugging among other things. This last stage took place during part of April and until the deadline, which means the start of July. This stage's work is showcased in the charts at 6.10.




**(a)** Hours per week in Stage 4




**(b)** Hours division in Stage 4

**Figure 6.10:** Total hours worked in Stage 4

### 6.5.1 Celery, RabbitMQ, Flower and Eventlet

In order to achieve the queuing system a combination of multiple technologies was used. Celery, would allow for a task distribution mechanism through queues and workers. Along with Celery, a message queue manager or broker was needed, so RabbitMQ was selected for this role as it was feature complete with Celery. And also for allowing concurrency within Celery, Eventlet would serve for running efficiently the workers by having them to process multiple tasks at the same time (although in the end it was not used as the default way for running the workers due to some issues experienced in the development, as expressed in chapter 7 Development), and finally in order to look at all of this functioning throughout, Flower, a web based tool that would allow to monitor all the tasks and workers and their related information and logs.

So through this combination a task queuing system was accomplished and was able to be integrated within the rest of the application. It is important to highlight that along with Celery a result backend is used for storing the returned results from the tasks, at first it was intended to use an independent application for this that also had support together with Celery, but in the end it was loadable to skip it and use one of the default ones that Celery brings integrated, which are AMQP and RPC, finally RPC was selected as the one to use, due to design reasons as it was explained in chapter 5 Design.

### 6.5.2 Web Sockets, Plotly and Jinja2

Finally towards having the last details and plotting in the task view, a charting library called Plotly was used which allowed also for real time plotting. This would be achieved along with the usage of the Web Sockets integrated within FastAPI, which is a technology that would permit the constant stream of data from the backend to the frontend in order to update the task's status and metrics charts. In addition, for hosting all of this, the Jinja2 template engine was used as it is a common election to use with FastAPI in order to provide HTML responses while returning data and showing it, although this was already used from the start of the application at the last part of the first stage, it wasn't until this one, where more detailed features from said template engine had to be used.

Ultimately it is important to point out that at first instead of Plotly, another charting library called Time Chart was used, which although for the first stages of the real time plotting served just fine, soon later it was found to be limited and lacking in features.

# 7 Development

As a follow up to the previous chapter where it was described how the development of the project was achieved, which guidelines and stages it went through as well as which technologies and tools were used. Now here, the explanation of the details regarding how all of these tools were used will take place. So in order to meet all the requirements wanted to be accomplished and satisfying also all of them so that they would work together as a whole system, how the mentioned tools were connected altogether will be portrayed.

To go through it in a way that is understood as easy as possible so that all the processes involved are explained but at the same time the whole pipeline is perceived, it will be explained in order based on how a user would normally create a request for training using MureTools, how it was developed and all the details implicated in its functionality.

## 7.1 Selection of models, parameters and corpus

First in order to train a model it was required to select one, so with the aim to let the user do so in our interface it was mandatory to expand the only form existent so that it could be used to create the same POST request for any of the included models that were relevant for our scenario in OMR.

Furthermore, due to the need of changing different parameters for different models and thus the necessity of hiding and showing different inputs, and also having the desire for them to be displayed even and treated as alike data so in the backend the same endpoint would supply the same services for all the models, as all of them needed the same processing. It was necessary to write the logic involving all of these changes through JavaScript functions that would adapt the inputs and let us know the wanted model to train and evaluate in the backend.

Also it is important to state that although originally it was desired and planned to allow for more customization in the model wanted through the possibility to change directly from the interface even the Artificial Neural Network (ANN) with its own parameters for each one, so that we could create our own models and experiment with them so the user could discover and compare results in different scenarios with different data, due to time limitations it was not possible to fully implement this feature although some of it is working within different testing endpoints, as its development started but had to be scrapped out, this feature is still preserved as it would be a really useful addition and also shows how much potential this could have, this is mentioned at a later stage in the Conclusions and Future Work chapter 8.

In addition to all of these specific parameters bound to a particular model, there is an input involving the corpus or data needed to train with, in our OMR case, it would be two

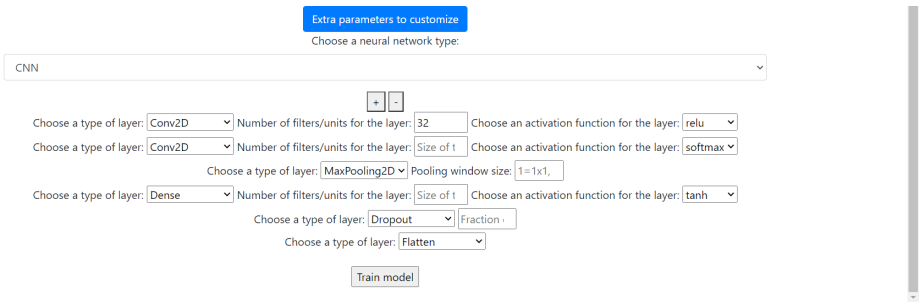



**Figure 7.1:** Capture from MURETOOLS showing the extra parameters section. It is visible all the different layers and also the different parameters related to the layer selected in each case

compressed files with all the respective directories containing the images files to feed the model and also the JSON files corresponding to these images, and that represent the ground truth for the images, or the actual results that our model should be able to predict.

Now once the request has been sent through this form, it will be received in the backend and now it is time for us to move on to the next stage in our pipeline processing.

## 7.2 Task creation, storing and start of training

Positioning ourselves in the backend of the application this time, firstly as the data passed could be inconsistent, a validation of the parameters was implemented so that not only the type of the data is checked but also the number of them in case of the corpus, as we expect two compressed files like it is shown in 7.1, FastAPI's in-built HTTPException serves as a great response for returning these messages.

Listing 7.1: Excerpt of code showing the creation of a FastAPI's HTTPException for validating the data received in the backend, in the case of the End to End model

```
@app.post('/train', response_model=Task)
async def run_training(corpus_files: List[UploadFile] = File(...), parameters: List[str] = File(...)):
    if not parameters[0]:
        raise HTTPException(status_code=418, detail="A model needs to be given!")
    else:
        model_parameters = {}
        if parameters[0] == 'CTC': #Parameters validation
            if not len(corpus_files) == 2: #Corpus validation
                raise HTTPException(status_code=418, detail="Two zip files are needed in order to train the CTC model! The json zip file and the images zip file corresponding to the corpus.")
            if not len(parameters) == 2:
                raise HTTPException(status_code=418, detail="There are parameters missing! CTC model needs a split number for dividing the training and test dataset.")
```

And once all the data is validated, it is converted to its due data type, as everything its received as an string, later on disposed in a Python dictionary and sent as a parameter to the Celery task initialization. Along with this also the path where the extracted files from the corpus compressed files are located will be sent, as a temporary directory will be created to host these.

As a result of the creation of this new training and evaluation task, an ID is returned from Celery's part, so in order to retrieve not only the status but also the resulting logs this ID

will be used. So an object of a Task class created in the backend in order to be used as a facade of Celery's tasks, with additional data, it is returned to the homepage as the response, being appreciated in 7.2. In the homepage all the requested tasks will appear underneath as they are created, although as the development advanced, for the sake of scalability and implementing new functionalities around the tasks, a new page only for the tasks was also created where it is possible to also filter them for example. Furthermore in order to have constancy of all of these tasks, they are stored in a list in the backend so every time we access the homepage they are retrieved and shown if there are any created during the current session. Originally it was intended to store them all within a database along with the results from said tasks, but due to time constraints it was not possible to create a unified database for the persistence of the created tasks in the backend with their related results, which are stored in the Celery in-built results backend, this is also mentioned in the chapter 8 Conclusions and Future Work.

Listing 7.2: Additional data returned in the /train endpoint, apart from the ID and status provided by Celery that are also returned

```
#Celery task facade
class Task(BaseModel):
    task_id: str
    status: str
    model_type: str
    date_time: str

@app.post('/train', response_model=Task) #The created task is returned
async def run_training(corpus_files: List[UploadFile] = File(...), parameters: List[str] = File(...)):
```

## 7.3 Implementation of models, training, evaluation and saving

Now in this section it is time to actually explain how the model requested is implemented underneath the system, and how the training, evaluation and subsequent request to MuRET is carried through, in order to upload the best model achieved according to the SER metric.

When the task is started through the endpoint, this task refers to one defined in the Celery tasks file, where through the model type parameter the desired model is executed, as all the neural networks available and machine learning related files are contained within this folder but on their own directory. In order to have a better picture on how all the files were organized, figure 7.2 depicts how all the relevant files and directories are distributed.

Then the rest of the parameters and the corpus directory path are sent to the function in charge of creating the specified model, also introducing these parameters in their respective spots so they are taken into account.

As a follow-up to the creation of the model, then the loop of training and evaluation takes place. Due to this metric used in OMR called SER, it was necessary to separate epochs so each call to the fit() function would mean a new unique epoch. After the training, the validation metric previously mentioned is calculated. This calculation is carried out through the Levenshtein distance, which is basically like an "edit" distance, so for example given two strings, the prediction just achieved and the ground truth, how many changes does the predicted one need to get to the ground truth or expected one, this way the deviation is quantified.

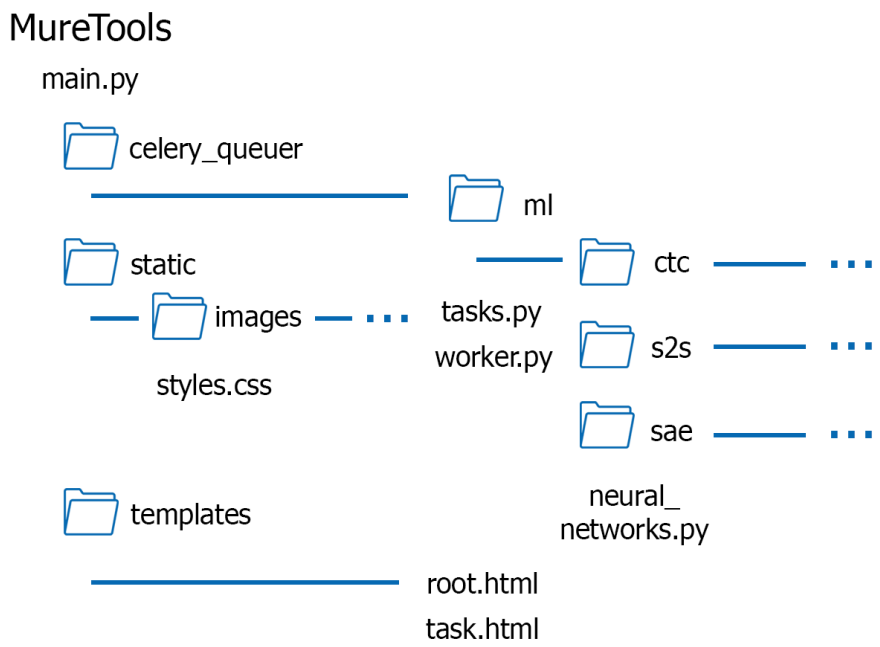



**Figure 7.2:** Directory structure with all the relevant files and directories of MURETOOLS. It is appreciated how the Celery queuer contains all the ML related files

Finally the totality of the metrics are added to the respective lists and the SER is compared to the best one stored until that very moment, if this results true then the new SER is stored as the best one and the model is saved in disk so the model with the best results can be later on sent through a request to MURET to be stored, so whenever is necessary the best model achieved is available.

In order to satisfy the need of displaying real time data for the user to follow along the training going on, it was necessary to use something that would allow getting the data as it was right after it was obtained through the fit() function, as once the training starts, it is carried out until it is finished. Here is when a Keras feature that allows for customized Callbacks comes into use, as during the training there are certain events like the end of a batch or the end of an epoch that are triggered and which behaviour is up to us to define. So this feature was used in this case for not only extracting the metrics at the end of each batch and epoch, but also for storing them into lists to be returned afterwards when the task is finished and to write into files so the Web Sockets could have a constant stream of the data needed by the charts.

As it was being developed, at first according to the requirements of being able to download a file with all the logs generated from the resulting metrics, everything was going to be dumped in files written in disk, so that it could be easily reached. Later on after trying

returning all of these metrics and logs through Celery's backend, this was decided as the way to go, so every time we download the logs and metrics from a task, first it is retrieved in its entirety from there and then written into files and zipped.

Furthermore, as in this whole process is necessary to create the vocabulary files in order to train and evaluate, and the model saved to be uploaded to MuRET along with the previously mentioned files, this arose the need of storing them all in a directory respective to the current task which after everything needing these files is complete the cleaning will take place so all of these files are not kept in disk after.

## 7.4 Logs and chart plotting

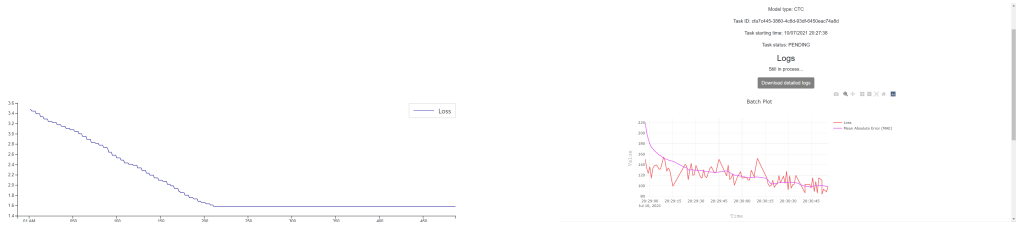
It was crucial for the development of MureTools to display the data to the user so that it would be easily perceived how the training and evaluation would advance as batches and epochs would success between each other.

In the last section it was mentioned how all the relevant data was registered as it was obtained through file writing to have an immediate availability and returning everything once the training and evaluation comes to an end. This is the reason why to make use of the pertinent data to be represented and maintain a constant stream of data, Web Sockets were used guaranteeing an ongoing connection between frontend and backend, so that this immediately obtained metrics could be represented in the charts. Moreover FastAPI permits the implementation of Web Sockets in the backend in a really convenient and straightforward way, so when developing this feature this was the preferred way to go as did not require any other technology or piece of software.

When it comes to the actual plotting and charting of the data, at first a really easy and simple library was used to only plot the metrics separately, called Time Chart as mentioned in the last section of chapter 6 Methodology. To guarantee first the functioning of everything as a whole it did its job really properly and could plot data in real time, but after using it for some time and seeing what it was capable of and its depth, it proved itself kind of lacking when it came to features, customization and overall use, so once everything was working altogether, the library was substituted.

After investigating and looking into libraries that could serve real time plotting the same way as well as offering more features, and trying other ones like Epoch (which seemed promising but turned out to be outdated and not working properly with the new updates of its dependent library D3), it was decided to use Plotly which was currently used and having active updates and community, which was preferred for maintainability and scalability. Through this library plotting different metrics in the same chart turned out to be fairly easy as well as customizing the different traces and the chart itself, also allowed for a more intuitive usage and introduced the possibility for the user to download the plot which was a really useful addition and saved work of developing something alike to satisfy this need. As soon as it was changed it made a huge difference for the better, as can be appreciated in 7.3.

When speaking of the metrics seen within the charts, although the relevant part would reside at the end of each epoch carried out, in order to perceive exactly what was going on during the training, a batch chart was also implemented. Once each batch finished the met-




**(a)** Time Chart chart **(b)** Plotly chart

**Figure 7.3:** Capture of a chart from both libraries, it is appreciated also the multiple options available at the top of the Plotly chart

rics would flow immediately to its respective chart, and analogically the same would happen for the epoch one. So by having the data regarding both ends, it is possible to find easily irregularities through the training and thus weigh how the overall training is doing.

Let's take as an example the following chart 7.4, where it is appreciated the SER metric of an End-to-End model training. Along with this, at the top it is partially perceived the batch chart showing the loss in red and the mean absolute error (MAE) in pink. So the loss and the mean absolute error although they have been steadily decreasing, the loss in red goes up and down around the mean absolute error, which just serves as an average line guide.


**Figure 7.4:** Chart showing the SER metric during a training of 20 epochs, the epoch average loss is hidden for readability

Despite this decreasing steadily, then in the SER metric it is appreciated that it is only at halfway through the training when it starts declining very rapidly, and not only that but there are times when the SER increases considerably, and before finishing the last minimum would have been the best model until that moment which would be saved and uploaded to MuRET, this is possible due to the way the training function was ran by only running one epoch and having a loop of the total of epochs desired to train, so the model saved is always the one related to that SER as seen in 7.3. The SER would have continued decreasing but it was a nice addition to also have the batch chart to contrast and visualize more globally everything.

Listing 7.3: Training loop seen at the end of the End-to-End training task

```
for global_epoch in range(20):#Originally is 50 iterations
    #Callback function to output the metrics per epoch for the user and each 5 epochs to be written in the logs json file
    logs = MyCallback()
    csv_logger = CSVLogger('training.log', append=True, separator=';')
    model_tr.fit(inputs,outputs, batch_size = 16, epochs = 1, verbose = 2, callbacks=[logs, csv_logger])
    ser = tuctc.getCTCValidationData(model_pr, X_val, y_val, i2w)
    #Metrics saved
    total_batch_logs.append(logs.batch_logs)
    logs.epoch_logs['epoch'] = global_epoch
    logs.epoch_logs['ser'] = ser
    total_epoch_logs.append(logs.epoch_logs)
    with open('metrics_epoch.json', 'w', encoding='utf-8') as f:
        json.dump(logs.epoch_logs, f, ensure_ascii=False, indent=4)

    total_pretty_logs.append('Finishing at '+str(logs.epoch_logs['end_time'])+' the SER for the epoch '+str(global_epoch)+', was '+str(ser)+', average loss was '+str(logs.epoch_logs['average_loss'])+' and the mean absolute error was '+str(logs.epoch_logs['mean_absolute_error']))
    if ser < best_ser:
        best_ser = ser
        model_pr.save("checkpoint_model.h5")
        print('SER Improved -> Saving model')

logs = {'epoch_logs': total_epoch_logs, 'batch_logs': total_batch_logs, 'pretty_logs': total_pretty_logs}
return logs
```

## 7.5 Queue system, broker and workers

According to what was designed and planned, MureTools required a task queuing system to manage all the requests of model training, since these would require some time to perform and meanwhile the user can still keep sending new tasks that will be taken and dispatched in its due time. During the implementation of this aspect of the project there were several issues to solve, not only how to manage the on ongoing training requests but also how to integrate everything with the architecture and technologies used in it, that is why it had to be researched upon in terms of functionality and how it really worked.

The base of how this works is the relation between broker and workers, so that the first one manages the message queue by getting each tasks created or published, and giving them by enqueuing it and making sure it reaches the right worker or consumer, then once a worker is available, it picks it up and starts executing it. In the case of needing to use multiple workers and threads as it could be selected, there is a possibility of doing so by using Eventlet together with Celery, so it was possible to provide an alternate execution pool implementation so that tasks can be treated concurrently and have multiple workers with multiple processes or threads, this way it would be possible to multi-process all of the accumulated tasks in the queues. This way of execution was tried but as sometimes while testing it the results were not exactly steady and there were some unexpected issues and due to the need of focusing on more crucial implementations, the execution sticked to the original single process one, this is also expressed in the chapter 8 Conclusions and Future Work. This is applied the same way to the case originally planned of the possibility in our system of taking into account the GPU current usage and capacity for a more efficient and flexible queuing system.

Nevertheless it is a nice addition in the future to have the possibility of working on it, as would open the path of more efficient processing per machine. Also once Eventlet is installed, it is really easy to change between execution methods as can be seen in 7.4.

Listing 7.4: Commands for starting the Celery worker in single (solo) and concurrency mode

```
Microsoft Windows [Versión 10.0.18363.1500]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Propietario\Desktop\TFG\MureTools>celery -A celery_queuer.worker worker --pool=solo -l
    ↪ info
C:\Users\Propietario\Desktop\TFG\MureTools>celery -A celery_queuer.worker worker --pool=eventlet -
    ↪ l info
```

In addition to all of this, although initially was also planned to have a more complex functioning involving measurement of the systems Graphic Processing Unit (GPU) usage destined to each task among other related stats, this was not quite consummated due in part to time limitations as well as other design and technologies involvements which were not taken into account from the very start. Since through this more refined behaviour it was expected to have the system to adapt to different scenarios so that it could decide more efficiently and have different reactions in response to the users' inquiries. Moreover this is also convenient in a larger context of scalability where MURETOOLS could be used jointly through multiple machines able to train models.

Focusing on the current scenario where a single user would create training tasks, as explained previously there are multiple ways that Celery can be configured and executed, and in the case of many configurations, instead of introducing them through the command line when executing it, they can be set up in the creation as shown in 7.5 or through a standalone configuration file.

Listing 7.5: Configuration of the Celery Worker seen in the worker.py file. Important to remark that the broker parameter is referring to RabbitMQ and the flag of persistence in the results backend is true, as by default it is set to false

```
from celery import Celery

app = Celery(
    'celery_app',
    broker='amqp://guest:guest@127.0.0.1:5672//',
    backend='rpc://',
    result_persistent=True,
    include=['celery_queuer.tasks']
)
```

To conclude, through its development it was useful being able to check the state and stats of the tasks at all times through Flower, a web based tool for monitoring and administrating Celery clusters, which would allow to check not only the tasks status and related information but also the workers carrying them out and all the detailed logs, as can be appreciated in 7.5.

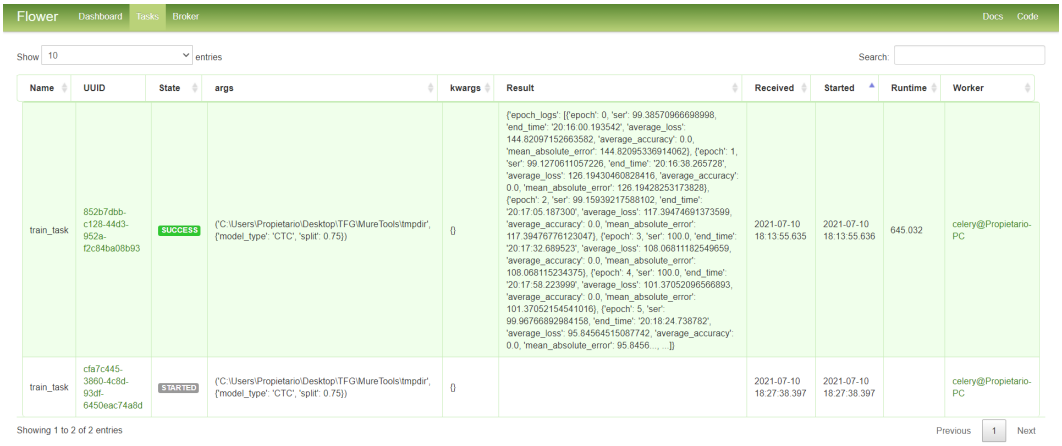



**Figure 7.5:** Capture from Flower showing logs and information related to the training tasks

# 8 Conclusions and Future Work

In this chapter, to summarize, all the proposed goals and final results will be weighed regarding also many of the difficulties faced, later also the many desired changes, improvements and next steps in this project will be expressed. Finally to conclude, the overall reached conclusions through the project and closing will be carried out.

## 8.1 Proposed goals and overall results evaluation

Although there were many objectives expressed within the Objectives chapter 3, to summarize all of them up, what was intended with MureTools, was a tool that integrated the many tasks within the OMR field into a convenient workflow for allowing training, evaluation and storing of models. To be concise, above all, the minimal viable product of such an application was desired, and we are able to verify its achievement through the weighing of the obtained results until now.

First and foremost MureTools possesses a way for the user to create requests training requests from a number of models provided, where they can also specify parameters influencing the model, as well as the corpus or dataset desired to train with. All of these selected models have been included with the objective of being able to have ways of recognizing and digitizing music scores and at the same time adapting to the data treated in OMR with its own formats as it is also explained. At the same time a validation of all of these parameters per model was successfully implemented and although there were many problems with the implementation of some of the models, specifically with the Musical Encoder and the Document Analysis ones, due to some technical issues related to the training and incompatibilities of the TensorFlow version, the GPU version of this library with the GPU employed in this project through my personal laptop which was outdated (NVIDIA GeForce GTX 1050) for today's standard in this field and its related computational processing load, and due to the whole situation propitiated by the outbreak of the COVID 19 pandemic, the possibility to go to the University of Alicante as well as using its equipment was truncated. Nevertheless it was possible to finally implement within our minimal viable product models that were relevant to the OMR field.

Along with this, a message queue manager was also implemented in MureTools so that these tasks could be managed in an organized and efficient way by taking place steadily. Through this method it was achieved the possibility for the user to request multiple consecutive requests and to also identify them using an identity to distinguish them and also be updated of their current status, so the user is able to know how the execution is going. Despite at first having the intention of implementing a more detailed system for managing all of these, that took into account more specific aspects like the current computation usage,

as well as the GPU capacity so more precise decision could have been taken depending on the running systems specifications, this objective was considered successfully satisfied as in a minimal viable product context, the mere fact of having a manager to take care of all of these asynchronous tasks is enough for considering it a success. Also, even though it was attempted to use concurrency within the system, there were troubles using it in some scenarios with the training tasks finally it was decided to stick with the single (solo) version of the queuing for the current application version. Altogether the possibility of this queuing system to scale as well as having investigated and weighed of such aspects serves greatly for bearing this in mind in the future.

Regarding the user interface, a usable and accessible one was intended. Also while being straightforward, simple and user-friendly and serving for the purpose of the user being able to carry out training and evaluation DL and more specifically OMR tasks and being updated on metrics and status of said tasks. All of these points were accomplished, even though there is always room for improvement and some details could have been improved as it is going to be explained in the next section, nevertheless this aspect was also satisfactorily achieved and without much problem.

So to sum up despite the inconveniences and setbacks experienced, and the deviation regarding the originally proposed objectives, the. So it is safe to conclude that the overall evaluation of the results with respect to the proposed goals, results in a successfully fulfilled and round project despite the shortcomings expressed previously and some others to be noticed in the next chapter.

## 8.2 Improvements and next steps

Now is time to declare some of the possible improvement and changes that MureTools could perceive which many were propitiated and inspired by many of the intended features that were planned from the beginning, but that due to time constraints along with other many complications, it was decided to scrap them out of the application for now in order to have a finished minimal viable product.

With the aim of expressing these improvements, they will be introduced in order of priority as to which of them would be the most crucial ones with the objective of adding the most value possible to MureTools, so to put it another way, the most essential features that MureTools would require to be at its ”best” originally planned version.

Firstly, a exclusive dedicated persistence system for the whole application that would contemplate not only tasks but also all the logs and metrics related to them a unified database where everything would sit together. As this has not been mentioned much throughout the whole thesis but it would be a very much needed feature, as it is perceived in most applications nowadays. Right now the results are stored within the in-built backend from Celery and the training tasks within our FastAPI backend, which makes everything clunky and not robust enough.

Secondly, the possibility of adding more parameters for the models implemented, as originally this was designed and even partially implemented but had to be scrapped out as mentioned in previous occasions. Also this would allow for scalability and even spark later the possibility of creating the user's own model by allowing even extra layers and parameters for said layers among other possible things.

Thirdly, the implementation of GPU computing and capacity of the systems working with MureTools within the message queue manager, as the one that MureTools is using right now is fairly simple in behaviour and only focuses on one task at a time and once it is finished then the next task is taken and so on, which means in the end only a single queue was employed in the current application version. So to conclude, there is plenty of room to improve in this aspect.

Lastly, improvements on the frontend that would improve the overall interface employed, like beautified validation messages in the form sending the training requests, more notifications, tools like modal windows could have been used, as well as creating a phone and tablet version of the interface, among many other improvements that the frontend could make use of, but as it is not as important as the previously explained features, it is expressed as the one that could be perceived more in the long run.

## 8.3 Final conclusions and ending

Ultimately, to wrap up this thesis, it is important to highlight the completion of a minimal viable product fully functioning within the context of providing supporting tasks in the OMR field. Furthermore through the development of MureTools many different concepts and fields were introduced and integrated under one single project, so it also serves not only as a tool for automating and creating a working pipeline within OMR, but also as a base and example of a project integrating all of these technologies and concepts, for the sake of further developments in the future supplying solutions for the needs that might appear in related fields.

# Bibliography


Andrés, M. R. (n.d.). Desarrollo de técnicas para el reconocimiento automático de manuscritos de partituras musicales. , 79.

*Basic Notated Music | Humdrum.* (n.d.). Retrieved 2021-07-03, from https://www.humdrum.org/rep/kern/index.html

Britz, D. (2016, January). *Attention and Memory in Deep Learning and NLP.* Retrieved 2021-07-02, from http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/

Brownlee, J. (2018, May). *A Gentle Introduction to k-fold Cross-Validation.* Retrieved 2021-05-23, from https://machinelearningmastery.com/k-fold-cross-validation/

Calvo-Zaragoza, J., & Gallego, A.-J. (2019). A selectional auto-encoder approach for document image binarization. *Pattern Recognition*, *86*, 37–47. Retrieved from https://www.sciencedirect.com/science/article/pii/S0031320318303091 doi: https://doi.org/10.1016/j.patcog.2018.08.011

Calvo-Zaragoza, J., Hajič Jr., J., & Pacha, A. (2020, September). Understanding Optical Music Recognition. *ACM Computing Surveys*, *53*(4), 1–35. Retrieved 2021-06-19, from http://arxiv.org/abs/1908.03608 (arXiv: 1908.03608) doi: 10.1145/3397499

Calvo-Zaragoza, J., & Rizo, D. (2018, April). End-to-End Neural Optical Music Recognition of Monophonic Scores. *Applied Sciences*, *8*, 606. doi: 10.3390/app8040606

Calvo-Zaragoza, J., Toselli, A. H., & Vidal, E. (2019). Handwritten Music Recognition for Mensural notation with convolutional recurrent neural networks. *Pattern Recognition Letters*, *128*, 115–121. Retrieved from https://www.sciencedirect.com/science/article/pii/S0167865519302338 doi: https://doi.org/10.1016/j.patrec.2019.08.021

Castellanos, F. J., Calvo-Zaragoza, J., Vigliensoni, G., & Fujinaga, I. (2018). DOCUMENT ANALYSIS OF MUSIC SCORE IMAGES WITH SELECTIONAL AUTO-ENCODERS. , 8.

Choi, K., Hawthorne, C., Simon, I., Dinculescu, M., & Engel, J. (2020, June). Encoding Musical Style with Transformer Autoencoders. *arXiv:1912.05537 [cs, eess, stat]*. Retrieved 2021-05-23, from http://arxiv.org/abs/1912.05537 (arXiv: 1912.05537)

*Connectionist Temporal Classification.* (2018, September). Retrieved 2021-07-06, from https://machinelearning-blog.com/2018/09/05/753/

*Convolutional Neural Networks (CNNs) explained.* (n.d.). Retrieved 2021-07-05, from https://deeplizard.com/learn/video/YRhxdVk_sIs

Dertat, A. (2017, October). *Applied Deep Learning - Part 3: Autoencoders.* Retrieved 2021-07-07, from https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Dinh, K. (n.d.). *An Overview of Microservices Architecture.* Retrieved 2021-07-06, from http://khoadinh.github.io/2015/05/01/microservices-architecture-overview.html

Dugar, P. (2019, November). *Attention — Seq2Seq Models.* Retrieved 2021-07-07, from https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263

Graves, A., Fernandez, S., Gomez, F., & Schmidhuber, J. (n.d.). Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. , 8.

IEEE. (1998, Oct). Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, 1-40. doi: 10.1109/IEEESTD.1998.88286

Inesta, J. M., Rizo, D., & Zaragoza, J. C. (2019). MuRET as a software for the transcription of historical archives. , 23.

*Intersection over Union (IoU) for object detection.* (2016, November). Retrieved 2021-05-23, from https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25* (pp. 1097–1105). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Kutvonen, A. (2020, July). *Get started with deep learning OCR.* Retrieved 2021-05-23, from https://towardsdatascience.com/get-started-with-deep-learning-ocr-136ac645db1d

Luong, M.-T., Pham, H., & Manning, C. D. (2015, September). Effective Approaches to Attention-based Neural Machine Translation. *arXiv:1508.04025 [cs].* Retrieved 2021-05-23, from http://arxiv.org/abs/1508.04025 (arXiv: 1508.04025)

Mishra, A. (2020, May). *Metrics to Evaluate your Machine Learning Algorithm.* Retrieved 2021-07-09, from https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

*PrIMuS dataset.* (n.d.). Retrieved 2021-05-23, from https://grfia.dlsi.ua.es/primus/

Rathor, S. (2018, June). *Simple RNN vs GRU vs LSTM :- Difference lies in More Flexible control.* Retrieved 2021-07-05, from https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57

Rodríguez Fortea, M. (2021, June). Murk. Videojuego de terror en Unreal Engine 4.

Ríos-Vila, A. (2020). *ReadSco. Open-Source Web-based Optical Music Recognition tool.*

Ríos-Vila, A., Calvo-Zaragoza, J., & Rizo, D. (2020, October). Evaluating Simultaneous Recognition and Encoding for Optical Music Recognition. In *7th International Conference on Digital Libraries for Musicology* (pp. 10–17). New York, NY, USA: Association for Computing Machinery. Retrieved 2021-05-23, from https://doi.org/10.1145/3424911.3425512 doi: 10.1145/3424911.3425512

S, S. (2020, October). *Gradient Descent: All You Need to Know.* Retrieved 2021-05-23, from https://medium.com/hackernoon/gradient-descent-aynk-7cbe95a778da

Scheidl, H. (2021, May). *An Intuitive Explanation of Connectionist Temporal Classification.* Retrieved 2021-05-23, from https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c

Shaw, A. (2019, August). *A Multitask Music Model with BERT, Transformer-XL and Seq2Seq.* Retrieved 2021-05-23, from https://towardsdatascience.com/a-multitask-music-model-with-bert-transformer-xl-and-seq2seq-3d80bd2ea08e

Shi, B., Bai, X., & Yao, C. (2015, July). An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *arXiv:1507.05717 [cs].* Retrieved 2021-05-23, from http://arxiv.org/abs/1507.05717 (arXiv: 1507.05717 version: 1)

Shin, T. (2020, January). *An Intuitive Explanation of Gradient Descent.* Retrieved 2021-05-23, from https://towardsdatascience.com/an-intuitive-explanation-of-gradient-descent-83adf68c9c33

Sutskever, I., Vinyals, O., & Le, Q. V. (2014, December). Sequence to Sequence Learning with Neural Networks. *arXiv:1409.3215 [cs].* Retrieved 2021-05-23, from http://arxiv.org/abs/1409.3215 (arXiv: 1409.3215)

Synced. (2017, September). *A Brief Overview of Attention Mechanism.* Retrieved 2021-07-07, from https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

Thomae, M. E., Ríos-Vila, A., Calvo-Zaragoza, J., Rizo, D., & Iñesta, J. M. (2020). Retrieving Music Semantics from Optical Music Recognition by Machine Translation. (Accepted: 2020-10-26T11:44:41Z Publisher: Tufts University) doi: 10.17613/605z-nt78

Tomás Pérez, J. V. (2020). *Recognition of Japanese handwritten characters with Machine learning techniques.*

*What is Perceptron | Simplilearn.* (n.d.). Retrieved 2021-05-23, from https://www.simplilearn.com/what-is-perceptron-tutorial

# Acronyms and abbreviations list

| | |
|---|---|
| **AMQP** | Advanced Message Queuing Protocol. |
| **ANN** | Artificial Neural Network. |
| **API** | Application Programming Interface. |
| **CER** | Character Error Rate. |
| **CNN** | Convolutional Neural Network. |
| **CRNN** | Convolutional Recurrent Neural Network. |
| **CTC** | Connectionist Temporal Classification. |
| **DL** | Deep Learning. |
| **GPU** | Graphic Processing Unit. |
| **GRU** | Gated Recurrent Unit. |
| **GT** | Ground Truth. |
| **IoU** | Intersection over Union. |
| **JSON** | JavaScript Object Notation. |
| **LSTM** | Long Short-Term Memory. |
| **MAE** | Mean Absolute Error. |
| **ML** | Machine Learning. |
| **MuRET** | Music Recognition Encoding Transcription. |
| **NN** | Neural Network. |
| **OCR** | Optical Character Recognition. |
| **OMR** | Optical Music Recognition. |
| **RNN** | Recurrent Neural Network. |
| **SAE** | Selectional Auto-Encoder. |
| **SER** | Sequence Error Rate. |
| **WER** | Word Error Rate. |